Universiteit Leiden
The Netherlands

Opleiding Informatica

Predicting photovoltaic power output with deep learning

Rutger Berger

Supervisors:
Thomas Moerland, Koen Ponse

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

24/06/2024

Abstract

Over the last decades, the capacity of installed renewable energy resources has increased significantly. However, with the increasing deployment of these resources, new issues arise. The inconsistent nature of photovoltaic systems, among others, is a major challenge for grid operators and makes it hard to implement large systems into the grid. Accurate forecasting methods are essential to ensure efficiency and security. Forecasting photovoltaic (PV) power output is in fact a time series prediction task and different approaches exist to gather accurate predictions. Deep learning based regression models are among the most successful ones. However, there are limited systematic comparisons of methods or studies into the influence of different parameters on this performance. For example, the history window of the data may have an important role on the performance of the models. This thesis provides a review of different state of the art deep learning methods applied to the task of short-term photovoltaic power output prediction, with a prediction window of 1 hour. The random forest model, which serves as baseline, performs better than any of the studied models. This surprising result is not in line with most literature, where the deep learning methods often outperform the baseline model, including the random forest. It suggests that either the training data is insufficient, or the problem of photovoltaic prediction does not contain many non-linearity's and long-term dependencies. The benchmark of the different models gives useful insight into the applicability of each model and the effect of different parameters within the same setting, providing an extensive analysis for any of the suggested methods.

Contents

1 Introduction

Whereas the growth of global emissions in CO2 decreases, the yearly quantity of emitted CO2 is still rising. The world is not on schedule to stay below the critical point of a global temperature rise of 1.5°C and the United Nations have warned more than once that we are on the edge of an irreversible change [LDDC24]. Fortunately, the yearly ringing alarm bells have made the awareness of the effect of CO2 on the planet increase. Following agreements of Paris (2015) and Dubai (2023), governments, industries and individuals shift away from fossil fuels and choose to invest in green alternatives. There are noticeable increments in the deployment of renewable energy production. For example, the installed capacity of renewable energy resources has increased by 35% over the last 5 years in the European Union alone [DMFM20]. The noble and aspiring intention of the EU is to be climate-neutral by the year of 2050[1]. Even though it may not be enough yet, many actions are taken to reduce global warming [RDK+22].

However, with the emergence of renewable energy sources and the increasing deployment of these, new issues arise. A major issue is the inconsistent nature of renewable energy production [RDK+22]. The electric grid is designed in the era where traditional power plants ruled the electricity generation, which may be fossil, coal-fired or nuclear. The power plants were very consistent and commonly only the demand on the electric grid fluctuated. The only challenge was to make sure the grid could handle peak demands, usually on early evenings, when every citizen was at home and cooking [PBBJ14].

Unlike the traditional power plants, the power output of renewable energy sources can be very fluctuating. In this thesis, focus is laid on the production of solar energy, often referred to as photovoltaic (PV) power production. The energy production of solar panels usually follows a very well-known day-to-day trend. Yet, it can be very volatile due to changing weather conditions. It is a major challenge to adapt the electric grid to handle the inconsistent production of PV resources [SAAS22][TPA+17]. Especially during extreme weather conditions, peak loads can flood the grid and decrease grid stability [WXX+17]. Furthermore, peak times of PV production and residential needs most of the times differ, an effect also known as the 'duck curve'. On a normal day, the solar irradiation is the strongest during the midday. When, after work, people return home and the irradiation diminishes, demand on the grid increases while people cook or stream their favorite series. This leads to imbalances in the grid and increased costs of dispatching. Moreover, the well intended electrification of many devices (e.g., cars), leads to a congested electric grid. For example, in some areas in The Netherlands, citizens or companies cannot build new houses and offices due to the shortage in grid capacity[2].

Many actions need to be taken to make the grid more robust and reliable for the increasing production of and the increasing need for renewable energy. Accurate forecasting methods for PV production, among other resources, are essential. They are not only necessary to improve efficiency but also required to assure security of the electric

[1] https://climate.ec.europa.eu/eu-action/climate-strategies-targets_en

[2] https://nos.nl/artikel/2502217-stroomnetten-weer-vol-nu-in-den-haag-groningen-en-overijssel

grid [DMFM20]. This thesis compares possible approaches in improving PV forecasting accuracy.

So far, different forecasting approaches have been proposed. The current state of the art approach typically uses deep learning based regression models. The prediction issue is in effect a time series prediction problem. Usually, studies use datasets from a solar plant of which sufficient data is available and fit different (neural) models to predict the PV power output based on meteorological data. As found in a review of related studies by Gallardo et al. (see [GAG23]), results of the studies are sometimes inconsistent but most of all often incomparable, due to different used system capacities and the dependence of the accuracy scores on these. Furthermore, most of these studies did not extensively benchmark different models on the same problem. However, the trend is that recurrent and transformer based models are among the most successful in the task of short-term power output prediction. Among important aspects on the performance of time series prediction models can be the history window in the samples of data. The history length can have a major influence on the performance of the models, especially when dependencies of variables are on a long-term bases. For example, within long texts the context of words in the last sentence can be dependent on something mentioned in the very first sentence.

A remaining challenge is to systematically benchmark the performance of different model architectures. Moreover, there is often no explicit study into using different history windows for the proposed models, including the fixed window approaches (random forest, linear feed-forward). At last, there remains a need for a study using Dutch meteorological data as none of the found publications have systematically tested prediction models for PV output of Dutch systems.

This thesis provides a thorough analysis of performance of different models in short-term prediction of PV power output, using a prediction window of 1 hour. Accurate prediction methods are useful and necessary for grid operators, to ensure grid efficiency and safety. For this study, data is retrieved from the Royal Dutch Meteorological Institute, the TU Delft and the European Union. Initially, models were trained on a small dataset, consisting of 1 year of PV power output, however the small amounts of data can limit the neural implementations in performance. Therefore a larger dataset is gathered from the European Union (EU), providing a dataset which describes 10 years of PV production at the same site. Proposed models are a linear feed forward neural network, a recurrent Elman network, a long short term memory network and a transformer. In the experiments, the influence of the history time window within training and test data is investigated. Moreover, the effect of using different model hyper parameters (learning rate, hidden layer size) on prediction accuracy will be measured. All accuracy is measured over unseen data and all data originates from the Netherlands. Results of the proposed deep learning methods are eventually compared to a baseline random forest model.

Results are interesting as none of the recurrent models or the transformer show increased

performance compared to the linear feed forward network. On top of that, the random forest model outperforms all of the deep learning implementations. The larger dataset increases the performance of all methods slightly. Increasing history windows does not affect performance much, if at all. Using longer history windows often even decreases overall performance, especially with recurrent networks. A history window of 2 is often already sufficient to achieve high accuracy, yet including the last 4 hours seems the foremost option in general. Increasing hidden layer sizes of neural architectures is beneficial to improve accuracy. Smaller configurations often suffer from underfitting. However, this effect stagnates when using hidden sizes of over 1024. Best measured mean absolute errors (MAE) range from 36.0 Watts (the RF) to 71.0 Watts (the RNN/LSTM) with a capacity of 2.000 Watt.

Our analysis of different approaches on predicting power output for Dutch PV systems shows that the problem may not contain many non-linearity's and deep learning methods do not necessarily improve performance. Another explanation for the found results could be that our experiment setup is wrong, for example by excluding too much features. Still, forecasting can be done rather accurately by using methods which are easy to fit and tune, as the random forest model performs the best and shows relative error scores of 2%. The next paragraphs will be used to provide an overview of current work, followed by an introduction and motivation to the proposed methods. The following preliminaries and methodology sections further describe the used data and models in detail, providing the necessary calculations and model architectures. The experiment setup concludes the main body of this thesis, by describing the proposed tests for each model. The last sections give space for an interpretation, discussion and conclusion of results.

2 Related work

Improving forecasting accuracy of PV power output has gained large amounts of attention over the last years. There are different approaches to this problem. This section first discusses similar studies and approaches, after which it dives into the topic of forecasting time series using deep learning. This way, a motivation is given for the used experiment set-up and methodology.

2.1 Similar studies

Within most studies, the spectrum of methods varies between physical models, traditional machine learning methods and deep learning models. Moreover, the prediction horizon varies extensively. Some studies focus on predicting for a time span of 10 minutes, whereas others attempt to forecast longer windows, with studies forecasting up to 1 month ahead ([PMK+20], for example). Approaches of the physical modelling kind first try to predict weather conditions and base the predicted solar irradiance (or flux density) on these weather conditions. Examples of traditional machine learning methods are random forest models or support vector machines. The most popular approach is

using deep learning, in which recurrent architectures, long short term memory models in particular, seem to be superior in most studies [GAG23].

Different studies find successes using different models. For example, Yung Ju et al. ([JLS20]) propose an encoder-decoder architecture using self-attention. The study finds significant improvements compared to known deep learning implementations. Phan et al. perform a similar experiment with a transformer based approach [PWP22] and their results are in line with Yung Ju et al. In a different experiment, Zhou et al. ([ZZY+19]) study photovoltaic power output forecasting based on long short term memory (LSTM) networks, also combined with attention mechanisms - and significant improvements are noted. In most studies, traditional machine learning methods like support vector machines or random forest often serve as baseline models and are then outperformed by the provided neural-like architecture. Found mean absolute errors differ extensively, depending on study and system capacity.

Standardisation is hard, since studies focus on local data and often do not generalize their findings to other datasets. The accuracy metrics are dependent on the specific dataset and system's capacity which can make the scores differ extensively between studies. Moreover, studies take place in different settings with different climates, making a proper comparison even harder. Remarkable is that in a review of more than 60 studies about this issue, no significant improvements within the last 3.5 years have been noticed, which could be due to the "limitations imposed by the physics of the problem (accurate weather predictions), and the relatively short time span of this review" [GAG23]. Suggested for further studies is to focus on data and features selection, and the effect on performance using different configurations and models.

2.2 Time series forecasting and deep learning

As mentioned, the issue of predicting PV power output is in fact a time series prediction task. Formally speaking, a time series is a series of data points indexed in time order, usually a sequence of discrete-time data. Examples are temperature levels, stock indexes and in our case power output of photovoltaic panels. Time series forecasting is the process of using a model to predict future values based on historical data.

Machine learning (ML), a broad and long existing field, concerns the use of data and training of algorithms to make predictions. ML is applicable to many tasks and particularly useful for time series prediction tasks [BN06]. Current state of the art performances in this field are often achieved by using deep learning. Typically, neural networks are good in capturing dependencies in complex and large datasets and usually suit well on time series prediction tasks. With the growing amounts of data and available computational power, deep learning increases in popularity and accessibility. For lower-dimensional tasks, the linear feed-forward neural network is often used, which is further described in the methodology section. However, many complex combinations of layers of neurons exist.

Latest developments in deep learning show results of high standards in many different

areas. Many techniques are becoming increasingly available, such as high quality video generation, natural language processing or image generation. The foundation of recent developments lays in the power of the attention mechanism, as found by Vaswani et al. in 2017 [VSP+17], as well as the access to large sets of data and the increasing capacity of GPUs (models can use over billions of parameters). Vaswani et al. showed by developing the 'transformer' that with enough data available and using attention, performances in natural language processing tasks can increase significantly. The transformer showed significantly better results than other models at that time, with an architecture solely based on attention mechanisms. For example, the nowadays well-known ChatGPT is based on a Transformer-like architecture. Before the success of the transformer, recurrent networks were considered the state of the art methods concerning sequential tasks. In essence, recurrent models keep track of hidden cell states for each time-step and use previous hidden states as input for new time-steps, which enables them to capture contexts and dependencies in sequential data of undetermined length. Recurrent models are known to be able to capture non-linear dependencies of features, learn functions of arbitrary complexity and deal with complex patterns as saturation or exponential effects [BMK+17]. Such methods suit well on time-series prediction tasks. This study implements two different flavours of recurrent networks, namely the Elman recurrent network (RNN) and the long short term memory network (LSTM). Rather than the RNN, the LSTM is known for its improved capabilities in capturing long-term dependencies and serves as an improvement to the RNN. The scope of this study is to compare different deep learning architectures, which include the feed forward network, recurrent networks and (the encoder of) the transformer.

3 Preliminaries

This section describes some necessities for the process of forecasting. It explains the features in the dataset and describes the performed steps during preprocessing.

3.1 Data preprocessing

Three different datasets are used. Originally, the data used in the experiments is collected from (a) TU delft open source PV power databases[3] and (b) the "Koninklijk Nederlands Meteorologsich Instituut" (KNMI)[4]. The TU Delft dataset describes the PV power output from a residential power plant in Amstelveen (the Netherlands) in watts (W) for every 15 minutes over the year of 2022. The dataset of the KNMI describes 21 different meteorological variables measured in Schiphol (the Netherlands) per hour, over the year of 2022 as well. Locations are 7.0 km away from each other. The relatively small size of the TU Delft dataset may limit most neural networks in performance. Therefore, solar data is in a second experiment collected from a different dataset, given

[3] https://www.tudelft.nl/ewi/over-de-faculteit/afdelingen/electrical-sustainable-energy/photovoltaic-materials-and-devices/dutch-pv-portal/pv-power-databases

[4] https://www.knmi.nl/nederland-nu/klimatologie/uurgegevens

feature i $(t-n)$	...	feature i $(t-1)$	feature i	PV power output in W
$x_{i(t-n)}$	...	$x_{i(t-1)}$	x_i	y

Table 1: An outline of the preprocessed dataset. As the data is ordered by timestamp, history is introduced by aggregating shifted datsets with the original dataset.

by the European Union[5]. The EU data differs slightly from the TU Delft data set. Essentially, the data describes hourly PV power output from 2005-2015. However, the data is converted from measured solar irradiation to hypothetical power production of a PV system on the same location. The capacity and efficiency of the system can be determined and configured by hand. For this experiment, the capacity is set to a similar capacity of the TU Delft dataset's capacity (2.000 Watt). Efficiency is set to a full 100%, which ensures the system's capacity the model is trained on actually is 2.000 Watt. The origin of the EU data is Schiphol. The EU data can be combined with the KNMI data, which is available for the same period and same geographical origin.

Preprocessing the dataset requires matching datasets by the timestamp feature. As the KNMI dataset uses hourly data and the TU Delft quarter-hourly, the PV power output dataset is averaged per hour to match both datasets. Some models (linear feed-forward neural network and random forest) require history introduced within each sample, whereas others (the RNN, LSTM and Transformer) require sequences of samples instead. For the first, history is introduced by aggregating (a) the actual data with (b) shifted data repeatedly. The data is then split into training and test sets on a 80 / 20 proportion. The test set consists of sequences of length 100 (hours), spread throughout the dataset so performance is evaluated on different seasons and years. The preprocessed dataset for FNN and RF models look as shown in Table 1. For the deep learning methods, extra steps include scaling data (to enable the models to learn faster) and for recurrent architectures to convert the data to batches of sequences with the history length as length of each sequence.

3.2 Feature selection

Features to be included are selected based on the calculated Pearson's correlation coefficient (Equation 1).

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{1}$$

The r is the correlation coefficient, determining how much the label is linearly correlated to the feature on a scale from -1 to 1. Next, x_i is the value of the x-variable (the feature) in the dataset, indexed by i. The $\bar{x}$ is the mean of the values of that same x-variable. y_i is the value of the y-variable (the label) in the dataset (indexed by i), which is the PV production in W and $\bar{y}$ is the mean of the values of the label.

[5] https://re.jrc.ec.europa.eu/pvg_tools/en/

Variable	Feature	Correlation score
HH	**Hour of day**	**0,34**
FH	Wind speed	0,07
FX	Max wind blast	0,09
T	**Temperature**	**0,44**
SQ	**Duration of sunshine**	**0,48**
Q	**Global solar radiation**	**0,43**
DR	Precipitation duration	-0,11
RH	Hourly precipitation amount	-0,06
P	Air pressure	0,08
VV	**Horizontal visibility**	**0,35**
N	Cloud cover	-0,12
U	**Relative humidity**	**-0,56**
M	Occurrence of mist*	-0,08
R	Occurrence of rainfall*	-0,14
S	Occurrence of snow*	-0,02
O	Occurrence of thunder*	-0,02
Y	Occurence of ice formation*	-0,03

Figure 1: The calculated correlation coefficient (see eq. 1) of several different features with respect to the power output of solar panels. A score of -1 indicates a perfect negative correlation, a score of 1 a perfect positive correlation and zero means no correlation at all. Noticeable is that statistically speaking, the duration of sunshine and relative humidity have a higher correlation to PV power output than the irradiation. Variables marked with (*) are expressed as boolean values (0, 1), which can explain lower correlation scores. Features in **bold** are included for the prediction task.

Outcomes of most features in the KNMI dataset can be found in Table 1. Some variables were left out in this table. Excluded are the indicator of attendance at weather station (IX), weather code (WW) and mean wind direction (DD), as they all all use meaningless scales (360=north, 90=east, 180=south, 270=west, 0=calm, 990=variable, for example for mean wind direction DD). Features are included in the dataset based on correlation scores. This step is thought of to only include relevant features for the task, as seen in [ZZY+19]. However, the risk of using this metric is that non-linear dependencies are excluded. This may therefore have of a strong effect on the final performance of the neural networks.

4 Methodology

This thesis implements five different architectures, based on data from the KNMI, the European Union (EU) and the TU Delft. Models are evaluated on the holdout set of test data, which consists of intervals of 100 hours of unseen data, spread throughout the years so performance is independent on seasonal changes. Accuracy scores of all configurations are noted and training curves of neural architectures are examined per different configuration. This section describes the architectures of the implemented methods.

4.1 Baseline model

As mentioned, we use a random forest as baseline model. A random forest, originally proposed by Breiman (2001) [Bre01], consists of a collection of N randomized decision trees. Once a decision tree is generated, it bases its outcome on the values of the input features. A visualization would be Figure 2, where a very simplified (and unrealistic) decision tree bases its prediction of y on the values of input features x_1 and x_2. The random forest has shown to be extremely successful in many classification and regression tasks. In our regression problem, a multitude of trees is generated after which the output of all trees is averaged. The random forest works on the bagging principle, which involves training multiple models independently. For a more thorough review of this method, see [BS16]. The average of the multiple trees serves as the final output of the random forest. The model has no max depth, uses gini as criterion for splitting and uses a default number of estimators of 100, which corresponds to the number of trees in the forest.

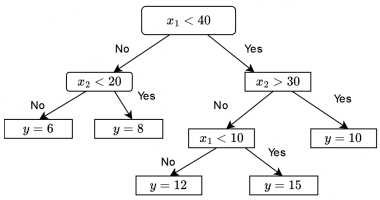

Figure 2: A very simplified visualisation of a decision tree regressor. Based on values of a feature, the root splits into new nodes and so on. In our case, x_1 could be temperature, for example. A random forest creates a multitude of trees (in an expanded version) and takes the average over all predicted values.

4.2 Feed-forward neural network

The fully connected feed-forward network (FNN) is a neural network, in our case built of three hidden layers and two drop-out layers (drop-out rate = 0.25). As activation function the ReLU function is used, so each layer calculates output by the following equations.

$$\phi(x) := \text{ReLU}(x) = \max(0, x) \tag{2}$$
$$h_1 = \phi(W_{h1}x + b_{h1}) \tag{3}$$
$$h_2 = \phi(W_{h2}h_1 + b_{h2}) \tag{4}$$
$$h_3 = \phi(W_{h3}h_2 + b_{h3}) \tag{5}$$
$$y = W_y h_3 + b_y \tag{6}$$

h is a vector of size hidden dimension, x a vector of size input dimension, W are weight matrices with trainable weight parameters and sizes dependent on dimensions of the layers it connects. The first dimension of the weight matrix in the first layer therefore corresponds to the number of features (input dimension) and the output dimension of

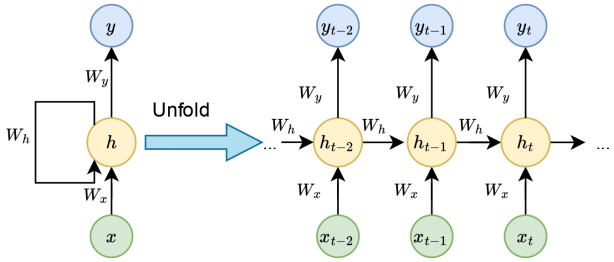

Figure 3: When the sequence of inputs is fed to the recurrent neural network (RNN), the unit repeats itself up till the last item of the sequence. Outputs of hidden states in the current timestep serve as input for the next timestep. When, for example, sequences of 4 hours serve as input, the RNN unfolds itself four times. The output of the last unit serves as prediction for the next hour. Image based on [RNN].

the last layer (y) is 1. The hidden dimensions are set by configuration and the effect of different dimensions is what we want to evaluate.

After each forward pass, loss is calculated using the mean absolute error (see Equation 7).

$$L = |y_n - \hat{y}_n| \tag{7}$$

In this equation, $\hat{y}$ corresponds to the predicted label. The final output of each forward pass is computed according to Equation 6, where the output of the last hidden layer is transformed to the still scaled label (as we have seen in the preprocessing section). Output is up-scaled reversibly to express output in Watts, so it is possible to express accuracy. Within the training process, mini-batch training is used, in which repeatedly small batches with a size of 16 are created. Within one epoch (a training cycle), the model repeatedly predicts one mini batch (one forward pass) after which the loss for each item is calculated (by Equation 7). The model updates its parameters using backpropagation. Backpropagation minimizes the loss by updating network's weights matrices and bias vectors. How much the weights adjust is determined by the gradients of the loss function with respect to each weight (by equations 8 and 9).

$$w = w - \epsilon \frac{\delta L}{\delta w} \tag{8}$$

$$b = b - \epsilon \frac{\delta L}{\delta b} \tag{9}$$

In Equation 8 and 9, $\frac{\delta L}{\delta w}$ determines the gradient of the loss function with respect to the weights in weight matrices W and $\frac{\delta L}{\delta b}$ for biases b. ϵ corresponds to the learning rate, which determines how much the model updates its weight each time. Values of the gradients are calculated using the chain rule (see [Wyt93] for a detailed outline of this process). After each epoch performance is evaluated by predicting the holdout set of test data, which is used to study learning curves and visualization of the predictions.

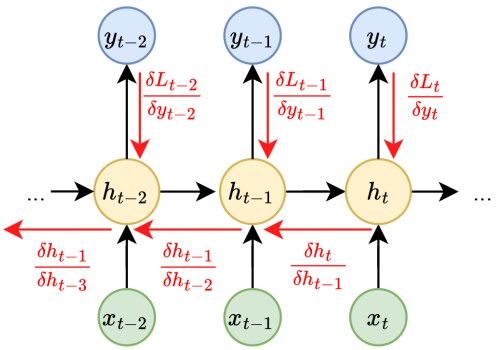

Figure 4: The intuition behind backpropagation through time (BPTT). BPTT unfolds a recurrent network in time, so it tracks output for every timestep. For every timestep, loss is calculated and summed. BPTT is used to find the gradient of the loss with respect to the parameters in the network. Image based on [Gup19].

4.3 RNN

As an extension of the feed-forward network, recurrent neural networks (RNN) are known for their ability in handling sequential data, such as sentences or time series. RNNs leverage hidden state vectors to capture short-term dependencies within sequential data, which enables them to perform complex tasks like machine translation where order is critical. Developed in 1986 (see [WZ89]), the RNN introduced the so called memory cell. The most common implementation is the Elman network (see [SM19]), which is used in this thesis as well. This thesis studies the performance of an implemented RNN after predicting PV power output based on the sequences of meteorological data. The RNN consists of one hidden cell, which recurs every time-step. The hidden cell state is updated each time-step, with the input features of the current time-step combined with the hidden cell state of previous time-step as input. The hidden cell then serves as input to the output layer as well as to the next hidden state in time (Equations 11 and 12). When feeding sequences, the RNN unfolds itself in time and repeatedly uses the same weight matrices to calculate next outputs, a process visualised in 3. This time, the Tanh function is used as non-linear activation function, keeping output between 0 and 1.

$$\phi(x) = \tanh(x) \tag{10}$$

$$h_t = \phi(W_x x_t + b_x + W_h h_{t-1} + b_h) \tag{11}$$

$$y_t = W_y h_t + c \tag{12}$$

h_t is the hidden state vector, of a size to be determined by configuration, W_h, W_x, W_y are weight matrices of dimensions (hidden dimension, hidden dimension), (input dimension, hidden dimension) and (hidden dimension, output dimension) and b and c are trainable bias vectors with a length equal to the hidden dimension and output dimension respectively.

Error is again calculated as mean absolute error. To update weight matrices and biases, backpropagation through time (BPTT) is used. During backpropagation, the error

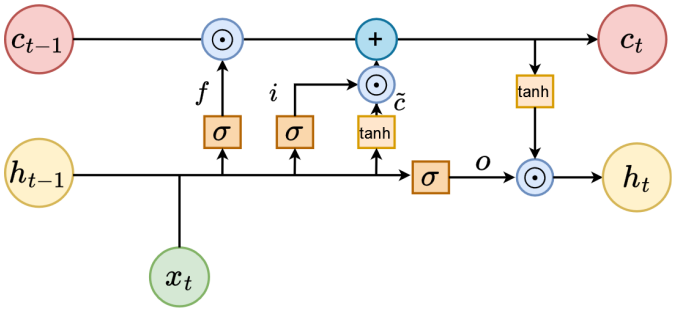

Figure 5: The architecture of the LSTM. The red c (cell state) on top ensures structure for long-term dependencies, whereas h (the hidden state) on the bottom acts like the recurrent cell from the RNN and corresponds to the short-term memory. Initial hidden and cell states can be chosen arbitrarily. Image based on [LST].

gradients flow backward through time, visualized in Figure 4. The unrolled RNN is now in fact a feed-forward network, but then with the same parameters repeated throughout the network, appearing at each time step. Just as in any feed-forward neural network, the chain rule can be applied, to backpropagate through the unrolled network. The gradient with respect to every parameter is then summed through all places the parameter occurs in the (unrolled) network. For further reading on backpropagation through time, I recommend the paper by P.J. Werbos et al., see [Wer90].

4.4 LSTM

Unfortunately, RNNs typically struggle in capturing long-term dependencies, as error gradients vanish when longer sequences are fed to the network, a problem commonly referred to as the vanishing and exploding gradient problem. In order to solve this particular vanishing gradient problem, long short term memory (LSTM) models introduce different sorts of memories and 'gates' to update the memories. LSTMs were initially developped in 1997 (see [HS97]) and many adaptations and improvements have followed (the PhD thesis of Gers et al. [GSC00], for example, introduces the forget gate). LSTMs usually perform better in capturing long-term dependencies in sequential data, which is useful for time-series prediction tasks such as ours. LSTMs are known as highly effective RNNs and are able to learn faster.

The implemented LSTM, much like the RNN consists of a recurring unit. However, the LSTM distinguishes long term memories and short term memories by creating two separate 'pipelines', as shown in Figure 5. Each time-step, the input is combined with the previous long-term memory ('cell state') as well as with the previous short-term memory ('hidden state') and both states are updated separately.

When sequences of data are fed, the LSTM unrolls just as the RNN. However, the LSTM consist of different memory cells which are calculated separately. Now, for every timestep, output is calculated and forward passes are done by performing different calculations,

as seen below. Figure 5 provides an intuition of what happens within the equations.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{13}$$
$$i_t = \sigma(W_i x_t + b_{xi} + U_i h_{t-1} + b_{hi}) \tag{14}$$
$$f_t = \sigma(W_f x_t + b_{xf} + U_f h_{t-1} + b_{hf}) \tag{15}$$
$$o_t = \sigma(W_o x_t + b_{xo} + U_o h_{t-1} + b_o) \tag{16}$$
$$\tilde{c_t} = \tanh(W_g x_t + b_{xg} + U_g h_{t-1} + b_{hg}) \tag{17}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \tag{18}$$
$$h_t = \tanh(c_t) \odot o_t \tag{19}$$

U and W are weight matrices but now of dimension (hidden dimension, cell state dimension) and (input dim, cell state dimension), respectively. As addition to the RNN, c is the cell state vector, which fulfills its role as long-term memory. The $\odot$ operator denotes the element-wise product of both vectors.

The hidden state at every timestep corresponds to the output of that timestep. Error is calculated the same way as with RNN and erorr gradients are determined by backpropagation through time.

4.5 Transformer

The current state of the art models in deep learning are very often based on the transformer architecture, as proposed by Vaswani et al. in [VSP+17]. For example, Chat GPT is in essence a gigantic pre-trained transformer, as the name already suggests (GPT = Generative Pre-trained Transformer) [WHL+23]. Transformers typically have an encoder-decoder architecture and make use of multi-head attention. In many sequence-to-sequence tasks, such as next-word prediction or translation tasks, using the transformer architecture has many advantages. Due to the multi-head attention, they are known to be very successful in capturing contexts within sequences and finding correlations between them. In this experiment, the encoder part of the transformer is taken and used for encoding the input of the meteorological variables.

4.5.1 Architecture

The proposed architecture uses the encoder architecture of the transformer and combines it with a feed-forward layer as decoder, see Figure 6. This way, input is encoded into a vector, using the advantages of attention and then decoded into the single feature of PV power output.
The multi-head attention requires input to be labelled with their position in time. To enable this, the first step in the encoding process is positional encoding. Positional encoding in transformers uses sine and cosine functions to create a fixed-dimensional vector for each item in the input. This vector captures the item's position based on its index. The positional encoded vectors are then passed to the multi-head attention layer.

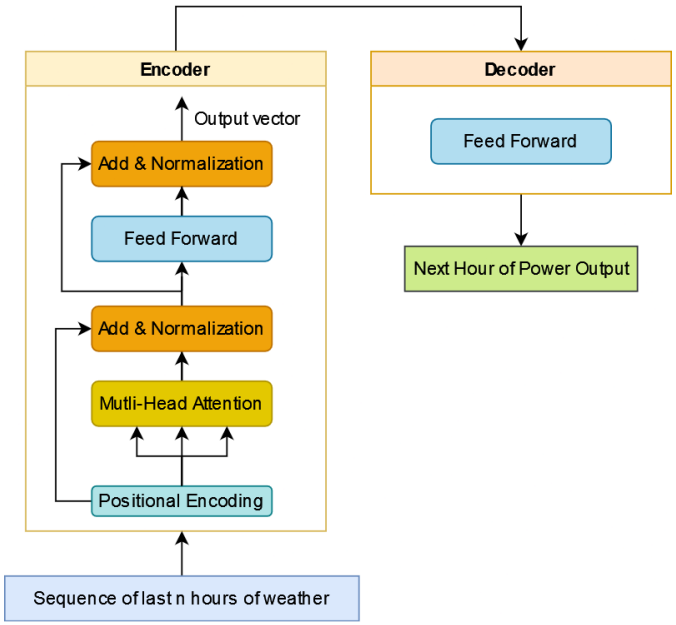

Figure 6: The architecture of the transformer encoder and used decoder. Each forward pass, input is transformed into an embedding vector (the output). This vector is then decoded, using a single feed-forward layer which converts the vector into a single number.

4.5.2 Multi-head attention

Multi-head attention increases the transformer's capacity to attend to specific information within a sequence. While traditional attention mechanisms focus on a single spot, multi-head attention divides this focus into multiple spots, by using multiple heads. The heads calculate the query, key, and value vectors assigned to each element of the input. Using multiple parallel attention matrices allows the model to simultaneously extract different relational features from the input data. The outputs are merged at the final step. The number of heads can be very influential to the performance, as an insufficient number hinders the capture of the different correlations, while a number too high can increase the computational costs.

Multi-head attention is a combination of multiple Scaled Dot-Product attention heads. Scaled dot product attention is defined by three different vectors, namely the query (Q) (what we want to pay attention to), key (K) (what our current input is 'offering') and value vector (V) (the vector we want to average over).

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{20}$$

Using scaled dot product attention, the model is able to assign different weights to different aspects of the input. However, there may often be different aspects (or: contexts) a sequence element can attend to. Therefore, multi-head attention increases the possible

number of 'contexts' by using more heads:

$$\text{Multihead}(Q, K, V) = \text{Concat}(head_i, ..., head_n) \tag{21}$$

$$\text{Where } head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{22}$$

Matrices W_i are trainable weight matrices which assist in defining the function of each vector.

4.5.3 Feed forward and normalization layers

After the Multi-Head Attention, a feed-forward network is added to the model, which is applied to every position individually and identically. For this feed-forward layer, the ReLU activation function is applied, identical to the proposed feed-forward network as seen earlier. However, the calculations are slightly different.

$$\sigma := \text{ReLU}(x) = max(0, x) \tag{23}$$

$$\text{FNN}(x) = \sigma(W_h x + b_h)W_2 + b_2 \tag{24}$$

$$x = \text{AddNorm}(x + \text{FNN}(x)) \tag{25}$$

The AddNorm layer is a way to normalize the distributions of the layers in-between, which ensures a smoother gradient and it lets the model train a lot faster. The feed forward layer increases complexity to the model and allows conversions on each sequence element individually.

4.5.4 Decoder

At last, when the input is transformed into a vector in the latent space, this vector is fed to the decoder. For our experiment, a single feed-forward layer is chosen as decoder. The layer is trained to transform the vector into a single item - namely the PV prediction. Output is calculated in the same way as in Equation 6.

5 Experimental setup

Different configurations will be tested for each model. The random forest architecture does not have many possible hyperparameters for its architecture and its implementation is quite straight forward. For the random forest model, performance is analyzed for different history windows. Unlike the random forest, the deep learning methods require many fine-tuning steps. Within this study, focused is on three options to adjust to increase performance: the learning rate, the hidden dimension and the history window. The learning rate, represented by ϵ in equations 8 and 9, determines how much the model's weights should adjust each time, based on the calculated gradients. In the experiment, the learning rate will be tuned with values ranging from 0.001 to 0.00001, dependent on hidden dimension size and network architecture.

Next, different hidden dimensions are possible for every architecture. Hidden dimensions are found in equations for the hidden states and hidden layers, see Equation 11, for example. Depending on the problem as well as the model's architecture, the performance of the model per hidden dimension can vary to a great extent. All models are first trained on the smaller TU Delft dataset, after which the best performing configurations are further trained on the larger EU dataset, which takes longer training time. For this experiment, models with hidden dimensions varying between 128 and 2048 are implemented. Dependent on the learning curve and results different ranges of dimensions are tested. Increasing the hidden dimension, however, has a negative effect on training time, due to the time-expensive process of calculating loss gradients with backpropagation.

At last, this thesis studies the effect of the history window of features by training all models on different history windows, varying from 2 to 10 hours. All model configurations are trained until convergence or with no further expected increments in test accuracy. Each training epoch, the models are evaluated by predicting the unseen holdout set of test data. Finally, for every history window in prediction samples, performance is noted for each configuration (which thus can differ in hidden dimension and learning rate). The transformer is set to be fixed at a hidden dimension of 2048 and for this case only the history window differs.

5.1 Accuracy metrics

To measure accuracy, the mean absolute error (MAE) and the root mean squared error (RMSE) are calculated. Both are known to be robust estimators of prediction accuracy for time series forecasting problems [DH06].

$$\text{MAE} = \frac{\sum_i^n |y_i - \hat{y}_i|}{n} \tag{26}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{27}$$

The lower both metrics, the better. The MAE calculates the average over all absolute errors, indicating absolute accuracy. The larger the difference between the RMSE and the MAE, the greater the variance of the individual errors is. The RMSE in its turn is more sensitive to outliers. As the errors are squared before they are averaged, the RMSE relatively assigns a heavier weight to greater errors. As we want all but many large errors or outliers, RMSE is considered a useful metric.

6 Results

Table 2 shows an overview of both MAE and RMSE scores for all of the proposed models in Section 4. The table describes how different models perform on the same test data using the two different datasets. The results tell us that the (baseline) random forest model performs better than any other of the proposed deep learning methods. The MAE and RMSE scores of the random forest model are, respectively, 36.0 and 75.2 on the large dataset. Based on a system capacity of 2.000 Watt, this indicates a relative error of 2%. Most methods perform slightly better when using a larger dataset to train, except for the feed-forward network. The feed-forward network performs nevertheless best out of the deep learning methods, followed by the RNN, LSTM and at last the transformer. Noticeable is that, for every deep learning model, training scores differ extensively from test scores. For example, with the RNN, training MAE scores could lower down to 15 with test scores of 70.

Compared to the random forest, the FNN yields slightly less accurate error scores. The MAE and RMSE scores of FNN's best noted performance are 42.5 and 85.9, using a history window of 8 (compared to 36.0 and 75.2 of the RF). On both datasets, the random forest does a better overall job than the FNN, yet with a small history window (≤ 2) performance is comparable. Noticeable is the decreased accuracy of the FNN when using the larger dataset. Perhaps the larger dataset contains more variation in unseen data, which could be a problem for the FNN. Furthermore, the RMSE scores are significantly higher than the MAE scores, for both methods. This indicates that both models lack in capacity of capturing outliers, corresponding to extreme weather events and sudden changes in weather. At last, the training time of the RF can take up to a few minutes whereas the FNN could require a few hours up to an entire day to converge to optimal performance. The random forest is therefore in advantage in both terms of accuracy and computational costs.

Following Table 2, we can see that the RNN performs comparable to the LSTM. The smaller dataset is in advantage of the LSTM, whereas with the larger dataset, the RNN is supreme. Best found MAE and RMSE error scores of the RNN are 70.2 and 77.3 and for the LSTM 71.2 and 79.1. The recurrent models have trouble in minimizing errors and capturing dependencies, with both methods performing poorly compared to the baseline and the FFN. However, both recurrent models are remarkably better in terms of handling outliers, since the MAE and RMSE scores are rather close to each other and the RMSE is even lower than that of the FNN.

The transformer performs the least of all methods. This method is only trained on the large dataset, which limits the possibility to perform a complete benchmark. However, comparing with the large dataset alone, the transformer shows MAE scores of 79.1 and RMSE scores of 92.8. Both MAE and RMSE scores are far behind compared to other model's and the model does not serve as an overall improvement to any of the methods.

Optimal performance of models using different history windows				
Model	Dataset	N (history)	Best MAE	Best RMSE
RF	S	0	88.1	165.7
RF	S	1	66.7	125.7
RF	S	2	44.3	89.6
RF	S	4	45.4	91.0
RF	S	8	43.9	86.7
RF	L	4	**36.0**	**75.2**
RF	L	8	37.0	76.6
FNN	S	2	44.3	91.6
FNN	S	4	44.0	88.7
FNN	S	6	42.8	88.8
FNN	S	8	42.5	85.9
FNN	S	10	45.7	90.2
FNN	L	4	50.6	101.4
RNN	S	2	93.1	102.1
RNN	S	4	85.7	105.4
RNN	S	6	86.3	118.6
RNN	S	8	86.5	124.9
RNN	S	10	87.3	132.4
RNN	L	2	70.2	77.3
LSTM	S	2	102.5	112.5
LSTM	S	4	89.0	110.3
LSTM	S	6	82.3	113.9
LSTM	S	8	82.2	120.3
LSTM	S	10	82.7	128.6
LSTM	L	8	71.2	79.1
Transformer	L	4	79.1	92.8
Transformer	L	8	79.4	106.2

Table 2: The performance of the different models and configurations. For example, for N = 2, the dataset includes the current hour and the previous two hours of features. The performance of the random forest is averaged over 50 stochastic runs. Noticeable is the almost unchanged effect on accuracy scores after increasing N up to more than 2. Features from over two hours ago do not seem to affect the accuracy for the next-hour PV-production. The scores itself are less accurate compared to comparable implementations [JLS20]. Moreover, none of the deep learning methods performs better than the random forest.

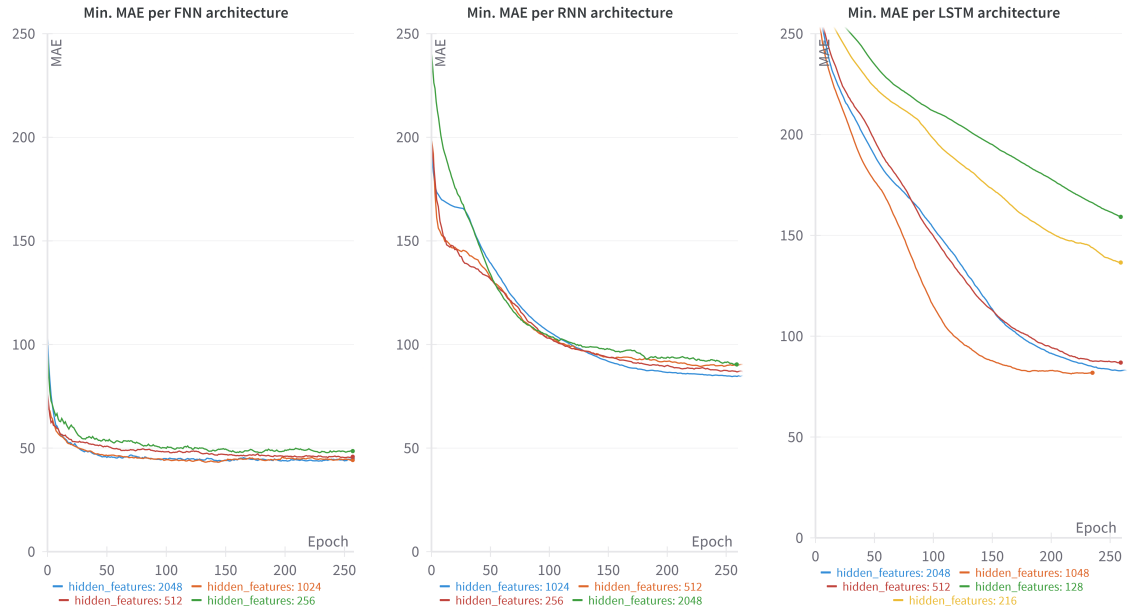

Figure 7: The learning curves of different configurations of the models when predicting on the holdout test set. From left to right, curves originate from the feed-forward neural network, the recurrent neural network and the LSTM. Per hidden dimension, the learning rate and history window is varied and thus each line describes the best performance per hidden dimension of the network. Used learning rates range between 0.001 and 0.00001, dependent on model size and performance. Training is done for around 250 epochs and for time related convenience stopped when no further improvements are found on the test set. The model's hidden dimension can have limited impact on the training curves. Whenever the hidden dimension is set too low, the model underfits and fails in finding relationships between features, which is also why the smaller LSTMs were stopped training before they converged. Optimal performance is often found using a hidden dimension of 1024.

6.1 Model configurations

In Figure 7, learning curves are shown for different model architectures. The figure shows the learning process for the FNN, RNN and LSTM using different hidden layer sizes for each model. Best performances are often found with a hidden dimension of 1024. This experiment excludes the transformer, for which limited time was available to benchmark the model for every configuration. The plots tell us that choosing different hidden dimension sizes can have limited impact on the learning process. For every model, the final performance often converges to a limit after which no further improvement is found. This means that there is a limit on how beneficial increasing the hidden dimension can be. The smaller hidden dimensions (256, 512) show slower learning curves with worse final performance and indicate an underfitted model. As mentioned, the hidden dimension of 1024 shows optimal performance with fast convergence. A hidden dimension of 2048 does not increase performance any further and only increases the required amount of training time. This can take up to 7 minutes per epoch instead of

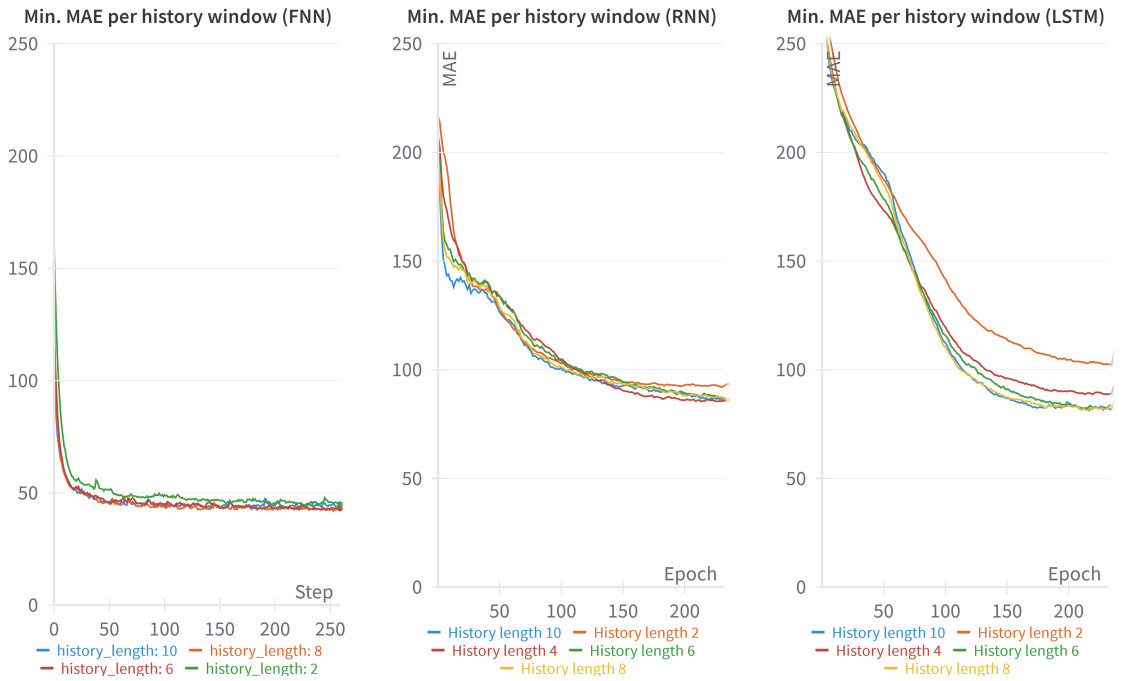

Figure 8: The learning curves of models on different history lengths. From left to right, plots correspond to the feed forward network, the RNN and the LSTM. Per history window, the learning rate and model architecture is varied and thus each line describes the best performance per history window. Training is done for around 250 epochs and for time related convenience stopped when no improvements are found on the test set. The history length can have a minor influence on training curves, yet which length is optimal differs per metric and per model. Especially the LSTM is sensitive to the history window, whereas the FNN and RNN do not show significant improvements. This may be explained by the focus on the long-term dependencies of the LSTM.

2.5 when using 1024. Another interesting result is that the required epochs to fit the RNN and LSTM is much more than that of the FNN. In terms of computational costs, the FNN is superior.

6.2 History lengths

Figure 8 shows the training process and performances of the deep learning methods for different history windows. The learning curve of the transformer can be found in a separate figure, see Figure 9. The figures and Table 2 tell us that the effect of increasing history windows after a certain level is marginal, yet differs for every model. For the random forest model, accuracy doesn't increase much after incrementing the history length up to two hours. However, the first two hours are essential. Performance increases from 88.1 to 44.3 (MAE, RF). Neural networks and especially recurrent neural networks show slightly different behavior. For the feed-forward neural network, a history length of at least four is required to ensure optimal behaviour. For the RNN accuracy decreases

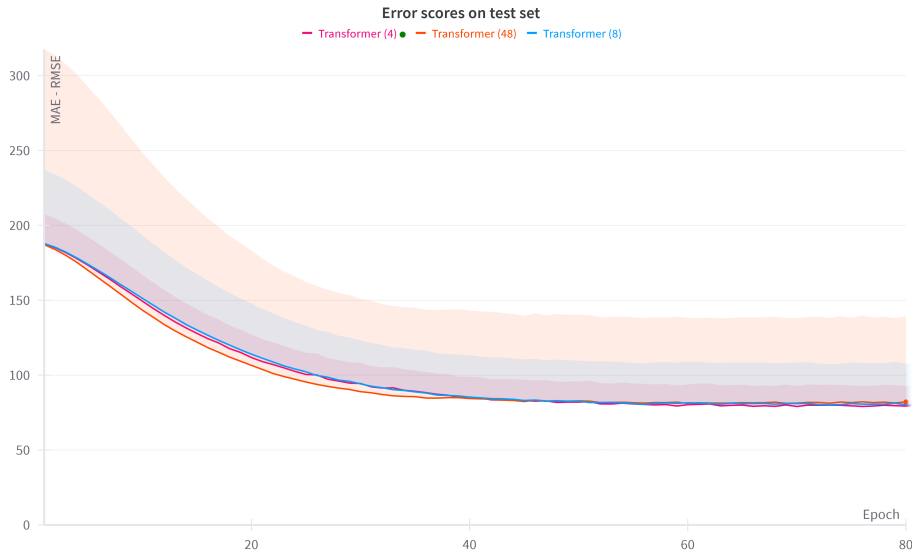

Figure 9: The training process of the transformer when using different history windows. The lower bounds corresponds to the MAE score and the upper bounds to the RMSE scores. Using a history window of 4 delivers the optimal performance, with lowest MAE and RMSE scores.

after inserting more history than the past four hours, yet for the LSTM model MAE scores are slightly more optimal for higher history lengths. However, with the smaller dataset, the LSTM shows optimal performance when using a history window of 2. The LSTM shows significantly worse results when using small history windows. Moreover, unlike other implementations, the RMSE score for LSTMs is not always in line with the MAE score. RMSE scores appear optimal for a history length of 4, yet optimal MAE scores are found using a history length of 8.

6.3 Performances on unseen data

In Figure 10, performance is visualized over different unseen test intervals. The intervals originate from the small dataset and cover three different seasons: winter, spring and summer. The figure shows us mainly two things. First, the predictions are quite in line with the actual trend, for every model. The deep learning models are correctly trained and succeed in the task of finding relevant features in the data. Next, we can see that the RF follows the line very accurately, whereas the RNN finds it harder to match with abrupt changes. Moreover, the RNN has difficulty in maximizing the predicted power output, as in some intervals (almost every one except for the first), the predicted output is 50-100 Watts less compared to the actual output. The FNN shows comparable behavior to the RF, predicting on a correct scale and handling most abrupt changes and peaks.

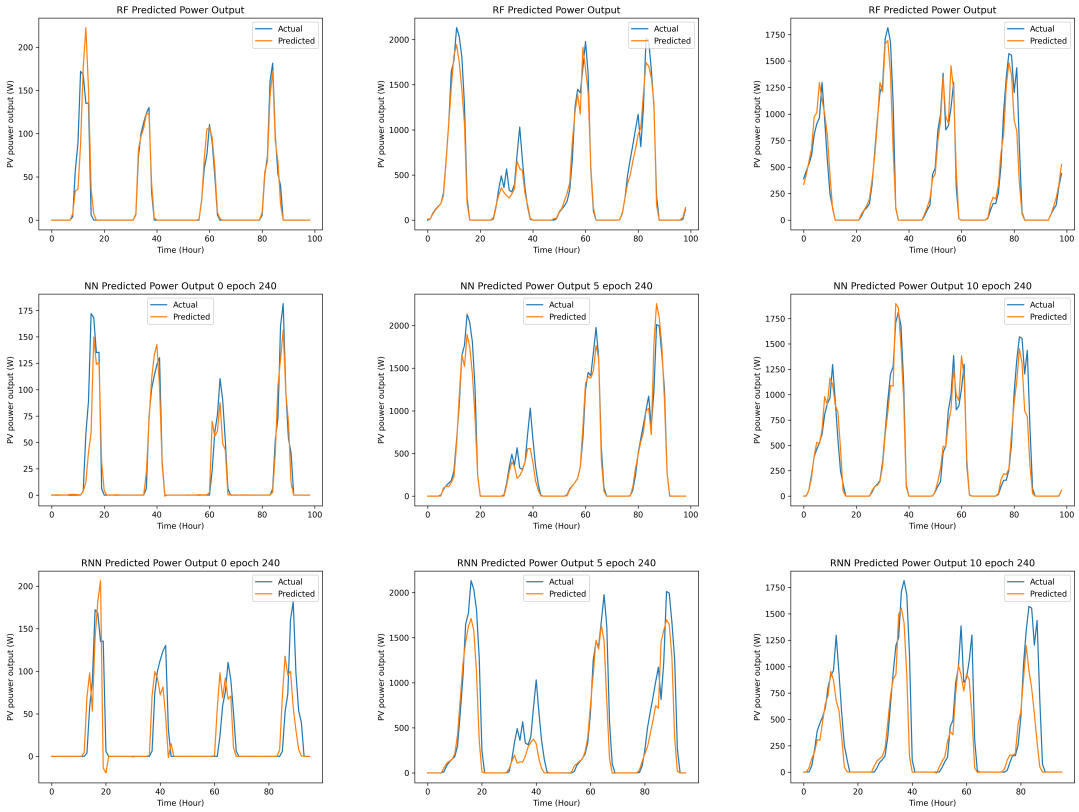

Figure 10: Performance for some of the different models when predicting three different unseen 100 hour-long intervals in the year of 2022. Output 0, 5 and 10 correspond to which batch in the testset is taken (the starting date of the interval). On top: the random forest (RF) model ($n_{history} = 8$), second row: fully connected neural network $n_{history} = 4$ and the last corresponds to the RNN, with a history window of 4. From left to right, intervals start at 13 January, 2 May and 1 August. Significant deviations from over 100 Watt at some points are shown. However, the overall performance is acceptable, with the predictions following the actual outputs neatly at most points.

7 Discussion

We find the random forest performing best out of the studied models. This result is not in line with most other studies, where the recurrent models in particular seem to be superior compared to most other models, including the random forest [DMFM20]. Furthermore, the relatively small impact of the history window is remarkable, in a way that it implies almost no long-term relations between variables. The combination of these two findings suggest that the proposed challenge of predicting solar output is a more plain and linear task than expected. In next two sections, I will give possible explanations for the findings and provide suggestions for further studies.

7.1 Limitations in the neural networks

A logical explanation for the initial bad performance of the neural networks is the limited available amounts of training data. The models initially only gather experience from 0.8 years of data from only one location, which could be significantly less than required. Indeed, the overfitting indicated by the the quite large differences in training and test loss suggest that the models need more data. The larger dataset increases the performance of most methods, however the error scores are still far behind compared to the random forest.

The recurrent models may not be very suitable for this specific experiment, for different reasons. For one thing, the difficulty in training RNNs properly is a problem acknowledged in other studies [Sut13]. The fine-tuning and fitting of the RNNs may have been insufficient to gather good results, even though the effect of many of the possible parameters has been studied. Second, the recurrent models especially suit for tasks with non-linear dependencies and changing lengths of sequences (sentences, for example). As mentioned, for this task, the dependency of the next-hour PV output on the given features seems to be only on a very short-term and fixed basis, with divergences on this function of only minor importance. For one thing, the RNN may have picked up signals as important indicators, which in fact may be noise and completely irrelevant. It is known that when temporal dependencies of data can be contained in finite and small time intervals, the usage of RNNs as well as Transformers may be unnecessary. In that case, methods using time-window approaches, such as the feed forward neural network (or even the random forest), are known to be better in terms of both accuracy and required computational resources [BMK+17].

Another reason for the limited performance of the neural networks may be the feature selection step in the experiment setup. Features are based on a linear correlation coefficient. A side effect is that this may throw away any feature with non-linear dependencies. My original thought was that this step was required to decrease complexity so the models would be easier to fit, especially the random forest. However, this is not in line with the strengths of neural networks. The power of neural networks lie in the fact that they can find (non-linear) relationships themselves and that they can identify important features. Throwing away some of the variables may have been problematic.

7.2 Future work

This study has investigated several factors for improving performance of different models, namely the dataset size, the history window and the model's hyperparameters. We find a clear overview of which method performs better in this setting. This can give useful insights into the applicability for any of the methods for this particular task. However, open questions remain. For one thing, as the random forest is already quite successful, we can wonder how high the need is for further studies into this issue. Following the results, the random forest may be sufficient for this particular task. The relevance of this study may therefore be less than expected. Moreover, we do not see many long-term dependencies in our results, as only the past two hours show a significant improvement in accuracy. Therefore, we can also doubt the setting of this task, on several bases. The relevance of the features for the predictor variable may be less than expected. Some excluded features could be states of the solar panels, or indeed the ones excluded in the feature selection step. Whether these give significantly better results, I doubt. One insight is that the PV-production is very much correlated to the solar irradiation. In fact, the PV power output of the EU dataset is a direct function from the solar irradiation. Predicting the solar irradiation, or at least, the weather, is something done for centuries. The local weather and thus the solar irradiation is dependent on more than the features currently included. Whereas in this study, we have regarded it as an stand-alone PV prediction issue, regarding the task as a weather prediction issue may perhaps show better results. Further studies could focus on actively using developed weather models to convert predicted features into a PV power output prediction, especially for long-term prediction. The results of this study suggest such an approach.

Furthermore, it remains an open question on how applicable the methods are in different (real-life) settings. A model with an error rate of 2% may be considered a very accurate predictor. However, whether grid-operators think the same is an open question. Additionally, the proposed models could be tested on unseen data from other origins, with other capacities, to investigate the generalizability of these models.

8 Conclusion

In this thesis, several different deep learning methods are compared to a baseline random forest model, on the task of predicting photovoltaic power output. The most successful method appears to be the random forest, instead of the expected neural networks. The model has a mean error of around 2% (MAE of 36.0 on a capacity of 2.000W) and outperforms all other methods, even when using a larger dataset. Moreover, the history window seems to be of almost no importance, as larger history windows only improve performance slightly. Optimal performance is often achieved using a history window of 8 hours, yet the positive effect of increasing this length is negligible after 4 hours. We conclude that the random forest model can be a good fit for this specific problem, due to the highest accuracy and the relatively low computational costs. Yet, we have also seen that the setting of this thesis can be flawed, due to the excluded features. Moreover, the high correlation of PV power output on solar irradiation and the limited

long-term dependencies we found suggest other approaches, for example, the use of weather forecasting models could be more successful. Upcoming hours of solar irradiation may be dependent on more than the features included in the data. The study into predicting PV power output is an ongoing process and this thesis provides an extensive benchmark of different methods on the same setting.

References

[BMK+17] Filippo Maria Bianchi, Enrico Maiorino, Michael C Kampffmeyer, Antonello Rizzi, and Robert Jenssen. Recurrent neural networks for short-term load forecasting: an overview and comparative analysis. 2017.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[BS16] Gérard Biau and Erwan Scornet. A random forest guided tour. *TEST*, 25(2):197–227, Jun 2016.

[DH06] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006. Twenty five years of forecasting.

[DMFM20] Aleksandar Dimovski, Matteo Moncecchi, Davide Falabretti, and Marco Merlo. Pv forecast for the optimal operation of the medium voltage distribution network: A real-life implementation on a large scale pilot. *Energies*, 13(20), 2020.

[GAG23] Isaac Gallardo, Daniel Amor, and Álvaro Gutiérrez. Recent trends in real-time photovoltaic prediction systems. *Energies*, 16(15), 2023.

[GSC00] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[Gup19] Pankaj Gupta. *PhD Thesis: Neural Information Extraction From Natural Language Text*. PhD thesis, 09 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[JLS20] Yun Ju, Jing Li, and Guangyu Sun. Ultra-short-term photovoltaic power prediction based on self-attention mechanism and multi-task learning. *Ieee Access*, 8:44821–44829, 2020.

[LDDC24] Zhu Liu, Zhu Deng, Steven J. Davis, and Philippe Ciais. Global carbon emissions in 2023. *Nature Reviews Earth & Environment*, 5(4):253–254, Apr 2024.

[LST] Wikipedia: Long short-term memory. https://en.wikipedia.org/wiki/Long_short-term_memory. Accessed: 2024-13-05.

[PBBJ14] Gareth Powells, Harriet Bulkeley, Sandra Bell, and Ellis Judson. Peak electricity demand and the flexibility of everyday life. *Geoforum*, 55:43–52, 2014.

[PMK+20] Debasish Pattanaik, Sanhita Mishra, Ganesh Prasad Khuntia, Ritesh Dash, and Sarat Chandra Swain. An innovative learning approach for solar power forecasting using genetic algorithm and artificial neural network. *Open Engineering*, 10(1):630–641, 2020.

[PWP22] Quoc-Thang Phan, Yuan-Kang Wu, and Quoc-Dung Phan. An approach using transformer-based model for short-term pv generation forecasting. pages 17–20, 2022.

[RDK+22] David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Sasha Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Kording, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer Chayes, and Yoshua Bengio. Tackling climate change with machine learning. *ACM Comput. Surv.*, 55(2), feb 2022.

[RNN] Wikipedia: Recurrent neural network. https://en.wikipedia.org/wiki/Recurrent_neural_network. Accessed: 2024-06-06.

[SAAS22] Md Shafiullah, Shakir D. Ahmed, and Fahad A. Al-Sulaiman. Grid integration challenges and solution strategies for solar pv systems: A review. *IEEE Access*, 10:52233–52257, 2022.

[SM19] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[Sut13] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.

[TPA+17] Vishnuvardhan Telukunta, Janmejaya Pradhaan, Anubha Agrawal, Manohar Singh, and S G Srivani. Protection challenges under bulk penetration of renewable energy resources in power systems: A review. *CSEE Journal of Power and Energy Systems*, 3:365–379, 12 2017.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wer90] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[WHL+23] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.

[WXX+17] Fei Wang, Hanchen Xu, Ti Xu, Kangping Li, Miadreza Shafie-khah, and João. P.S. Catalão. The values of market-based demand response on improving power system reliability under extreme circumstances. *Applied Energy*, 193:220–231, 2017.

[Wyt93] Barry J. Wythoff. Backpropagation neural networks: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.

[WZ89] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[ZZY+19] Hangxia Zhou, Yujin Zhang, Lingfan Yang, Qian Liu, Ke Yan, and Yang Du. Short-term photovoltaic power forecasting based on long short term memory neural network and attention mechanism. *IEEE Access*, 7:78063–78074, 2019.