Development of an AI based Load Prediction Algorithm and its Implementation into an Open Source Energy Management System

Nils Reiners
Electrical Energy Storage Division
Fraunhofer Institute for Solar Energy Systems (ISE)
Freiburg, Germany
nils.reiners@ise.fraunhofer.de

Eric Schüftan
Computer Science Department
University of Münster
Münster, Germany

***Abstract*— In this paper, two artificial intelligence based load prediction algorithms are presented, one of which achieves improved prediction performance compared to the standard method. It is also shown what results can be achieved when the algorithm is implemented in the open source energy management framework OpenEms.**

Keywords-artificial intelligence; neural network; load prediction; energy management system; openems

I. INTRODUCTION

Generally Energy Management Systems (EMS) are needed to optimize the operation of energy systems. This is true for grid connected as well as for off-grid systems. The implemented EMS strongly differ in complexity depending on the purpose and the size of the corresponding energy system. Simple EMS just control view components and are giving set-points for the current time step. In a system including a photovoltaic system (PV) and an electrical storage system (ESS) this can be the maximization of the PV self-consumption by charging the ESS if excess energy is available and discharging the battery when there is an energy deficiency.

More sophisticated EMS systems include scheduling algorithms. That means that the optimal setpoints for the components are calculated for a future period of time (e.g. 15 hours). The calculation of the future schedules can be updated continuously such that changing system states and updated predictions can be included. Such strategies are also referred to as Model Predictive Control (MPC).

To create a schedule for a future time period predictions are needed for the energy production and for the energy consumption of the system. The quality of these predictions determine the quality of the schedule. PV predictions are commercially available in a high quality and can be included in EMS platforms via APIs.

A reliable load prediction on the other hand is more challenging because every energy system is different in terms of components and the usage of these components strongly depends on user behavior and e.g. specific industrial processes. A reasonable approach is to use monitoring data to learn something about the behavior of the system. An often applied approach is to use a combination of the monitored data of the day before and from the same day in the last weeks and to calculate the load prediction from these values. But these simple approaches often have a low prediction accuracy. To receive better results we used two different approaches in this work to calculate the load prediction with the help of Deep Neural Networks (DNN).

We structured the paper as follows. We first describe the three approaches that we used for the load prediction. We then describe the example energy system for that these load predictions were tested. Subsequently we describe the software platform OpenEMS in which the load prediction was implemented. Then the simulation procedure is explained. Part of this implementation was also a procedure to calculate the schedule for the next 15 hours that will also be described here. Finally the results are presented.

II. LOAD PREDICTION METHODS

Two different DNN based load prediction concepts were developed that will be explained in the following. These were compared to a standard benchmark prediction method that is also explained in this section.

The dataset that we use for the training of both models consists of load data from 114 American single-family households and weather data. The households were metered from late 2014 - 2016 and contain load power in resolution of one or fifteen minutes. The weather data contains information about temperature, precipitation probability, cloudiness, visibility and other and was recorded throughout the same period and for the location of the households. The data set can be found online at [1], we choose this set because of its comprehensive size and public availability. Its resolution of 1 to 15 minutes is more than su_cient for our application as the load forecasting that we implement does not need higher resolution input loads than of one hour intervals.

A. NN-F

The first model is called NN-F, which stands for "Neural Network Forecast". The choices made for topology, training and hyperparameters are examined in the following.

Architecture

There are various neural network architectures possible for regression estimation like load forecasting. That includes feedforward neural networks, recurrent neural networks (RNN) and especially long short-term memory (LSTM) networks.

LSTM networks are specialized RNN that have proven powerful in recent literature for short-term load forecasting, too [2], [3]. Testing their aptitude for STLF without individual building training data remains open for future work. In this work, for practical reasons, we choose the standard feedforward architecture of ANN. The topology of NN-F consists of one input, one output and five hidden layers. The input neurons are listed in the table 1.

TABLE 1: SPECIFICATION OF THE INPUT NEURONS

	Number of input neurons
Loads of 7 last days	168
Loads of last 15 hours of same day 2, 3 and 4 weeks ago	45
4 weather inputs for the next 15 hours	60
Weekday and hour of day	2
Sum:	275

The hidden layers contain 150 neurons each. The output layer contains 15 continuous values that represent the 15 hours of forecast time span.

We choose this topology to allow the detection of complex relations between input and output while also keeping the complexity relative to the size of our data. Given that we train on a data set that contains single-family households that are located in relative proximity we have to take into account that we only cover a certain subset of consumer types. This reduces the representativeness of our data. We pretend that this subset of the problem of load forecasting is equal to the superset. On that assumption the complexity of our problem of consideration is reduced. This makes our model more vulnerable to overfitting, which is why we do not want it to look for too precise and most likely only individually valid correlations. The model depth is only a small contributing factor to the problem of overfitting, though. This has to be recognized in training, too.

Next is the choice of activation functions. In the output layer, we are restricted to continuous functions as we expect a continuous output value for each of the 15 output neurons representing the hourly loads. The ReLu function meets this criterion and furthermore limits the output to values equal to or greater null. This makes sense in the context of application as load values cannot be negative either. The activation functions of all other layers can be chosen freely. We opt for ReLu in all layers. Very important for the successful training and left to choose is the loss function. In testing different loss function we come to the conclusion that the common losses like MSE, RMSE or MAE do not fit our problem. These loss functions display the average deviation between discrete load values. This results in an approximation that has only few high discrete errors but many errors of medium height and only few sufficiently low errors. Therefore, the approximated energy volume is very accurate, but the approximated load curve is rather flat.

The flat load curve is undesirable because it does not display the momentary progression. To target this problem, we design the following custom loss function MSGE.

$$MSGE(y, y*, w) = (1-w)\frac{1}{n}\sum_{i=1}^{n}(y_i^* - y_i)^2 + w\frac{1}{n-1}\sum_{i=2}^{n}(y_i - y_{i-1}) - (y_i^* - y_{i-1}^*)^2$$

MSGE stands for "Means Square Gradient Error". It is a modified MSE that adds a term to capture the curve gradients more actively. Besides the average squared discrete errors it penalizes missing the change of gradient. We achieve this by calculating the mean of discretized first derivates. We weight both terms with w = 0.5.

Training

NN-F is trained on the full data set of 114 households and weather. Necessary before training is finding the right partitioning of training and test data set. It is desirable to use as many data samples as possible for the training of the model to fully utilize the data on the one hand. On the other hand, the test data share must be of sufficient size to allow a meaningful evaluation. Accurate evaluation of the model benefits the optimization process

of it. Therefore, we must find a reasonable compromise. We opt for a fixed

share of 5/6 of each household of the whole set for training data. This leaves 1/6 in each household for testing.

At this point it is possible to scale or standardize the data, e.g. map to a fixed interval. This can make sense if the data varies in scale. In our case, this is not needed. The actual training is performed in 8 full iterations of all training samples (epochs).

Implementation

We implement NN-F inside the machine learning framework Keras. Keras is a standardized Python interface for different machine learning libraries. In this case we use Keras with TensorFlow, an ML and particularly deep neural network library. See [4] for documentation of TensorFlow and [5] for documentation of Keras. This allows us to draw on a comprehensive set of implemented high-level functions that allow the design of neural networks.

B. P2NN-F

The second model that we propose is P2NN-F, which stands for "Profiler To Neural Network Forecast". This novel forecasting model aims to increase the accuracy further. It will be introduced in the following with focus on the conceptual choices that we make.

Concept

The basic idea of P2NN-F is to identify consumer types and assign new customers to one of these types before the forecast is made. Having classified a household correctly, we want to be able to create a more customized forecast, that is thus more accurate. For that we train a specialized ANN for each consumer type.

This idea is founded on the assumption that separable consumer types exist in small-size households. In practice there often form distinct groups of similar consumer habits and behavioral patterns of consumption, which is why user behavior can be categorized in many application fields. See [6] for an example about the clustering of the usage of refrigerators.

Architecture

P2NN-F consists of two parts, clustering and forecasting. In the first part we introduce the clustering. The forecasting is subject of the second part. And the choices of model hyperparameters are explained in the third part.

Clustering

To identify consumer types, we cluster the whole data set into k load profiles. As clustering algorithm we choose k-Means because it is both efficient and simple. Its simplicity allows an intuitive understanding and helps us optimize input choices and other impactful decisions in the context of the model. Kmeans is depicted in subsection 2.2.5. The inputs for this clustering are obtained as follows. We split the load data of every household into one day sequences. One such sequence is constructed as the mean of all 5 loads of working days in a week, from Monday to Friday. By that, we try to identify the typical working day consumption profile of the respective household. Weekend days are left out as they differ significantly from working days in many households.

In this approach we make assumptions that are based on simple everyday observations. This has to be kept in mind. For practical reasons, the averaged days are normalized. Each day is mapped to the interval [0; 1]. This ensures comparability between all sequences. A consequence of this solution is that every household contributes a number of averaged clustering days to the pool of clustering sequences. Therefore, a profile can contain sequences from many different or possibly all households. But we only want to take the data into the training data pool of a specialized ANN that represents the respective consumer type. For that reason we define a limit to take in only the first n households in a list ranked by contributions of sequences made to the profile.

To assign a new household to a profile, we consider 4 weeks of continuous loads. From that we obtain 4 averaged input days that we align to the previously clustered profiles. We then choose the profile with the most matches.

The choice of 4 weeks allows us to have multiple days to cluster while staying inside the time frame of one month needed before the model can be applied. This time frame is necessary for the forecasting, too.

Forecasting

The second part in P2NN-F is the actual forecasting. For each of the k profiles, we train a neural network. The topology of the networks is similar to NN-F with equal input and output vectors but only 3 hidden layers. We choose the model's depth smaller to pay respect to the fact that - for each individual ANN - the available training size is much smaller. The loss function and the activation functions are the same as in NN-F.

Choices of model hyperparameters

The first hyperparameter left to choose is the number of profiles or clusters k. We choose k = 6 for following reasons. With increasing k, the sum of squared distances is strictly monotonically decreasing until k equals the number of inputs. In that case it is null. Therefore we do not look for the k that minimizes the sum of squared distances but for the one that is the smallest possible that produces a sufficiently small sum of squared distances. The term sufficiently is - of course - relative and application dependent.

We identify the interval [4; 7] as the sweet spot of already lowering the sum of squared distances significantly compared to lower values of k and at the same not being significantly worse than higher ones. The other reason for choosing k = 6 is a graphical comparison of k's in that interval regarding their produced cluster centers.

We choose n = 30 for the following reasons. n should neither be too small so that the forecaster lacks data nor be too big so that the intersection between the data pools of all forecasters is too big.

In the latter case we run the risk of creating forecasters that generalize too much due to having too many different input loads of different types of consumers when optimally it would only have loads of the very same consumer type. That would lead to forecasters that are too similar to each other.

And this would contradict the formulated goal of implementing specialized forecasters fundamentally and must therefore be avoided at all cost. In the case that n is too small, another side effect is that there remain households that are taken into no data pool of a forecast. This reduces the eventual forecasting quality of the model potentially if we forecast loads of households that are not taken into training of any profile. With n = 30 we hope to find a compromise in that conflict of interests. For this choice of n we observe a representativeness score of little over 90%, which refers to the share of households taken into data pool of at least one forecaster.

Training

By the training of P2NN-F we refer to the training of the 6 forecasters that the model contains. For the household data of each forecaster we select the same training partition as in NN-F. This will important later on. Like in the training of NN-F, we do not need to scale the load input data. The training of P2NN-F is also performed in 6 iterations of the full training data pool.

Implementation

The framework for the implementation of the neural networks is the same as with NN-F. For k-Means, we rely on the Scikit-learn implementation of Lloyd's algorithm or k-Means. See [7] for details of the of the Scikit-learn library.

C. Benchmark Forecast SW-F

To compare the proposed models of this work, a benchmark model is needed. Such benchmark model is ideally both well performing and simple, that is easy to (re-)

implement. In [8], the authors give an overview and comparison of simple load forecasting models. Regarding MAE and MAPE, best performing is a statistical method that takes in the last k same weekdays of load, weights them and calculates their mean. Based on this, we implement a model that we call SW-F, which stands for same weekday forecast. To get a comparable output of cardinality 15, we expect an output in the form $y = (y_0; \dots; y_{23})$. Let loads of the same week day n weeks ago be $x_n = (x_{n0}; \dots; x_{n23})$, weights $w = (w_1; \dots; w_k)$ and all considered same weekdays together be $x = (x_1; \dots; x_k)$, then we define

$$SWF(x) = (y_0, \dots, y_{23}), y_i = \frac{1}{k}\sum_{n=1}^{k} x_{ni} w_n$$

SW-F makes use of the strong correlation between past same weekdays and the day of forecast. Additionally, all days considered are weighted. This comes from the fact that not every past day has the same inuence to the day of forecast. To implement SW-F, we have to start by choosing an according K. As we previously chose the maximum range of consideration of past days to be of 28 days, we choose K to be 4. This means we take days of 7, 14, 21 and 28 days ago into consideration. Lastly, we have to choose weights. Usually, more recent days have a higher inuence on the day of forecast than later ones. To express that observation, we choose w = (0.5; 0.2; 0.2; 0.1). Our benchmark forecast model is thereby finished.

III. SYSTEM SETUP AND SIMULATION CONCEPT

On the one hand we want to evaluate the quality of the before presented load prediction algorithms and on the other hand we want to see how they perform when they are implemented into a real EMS. The first part is done by comparing the two neural network predictors to the benchmark predictor.

For the second part a simple energy system was chosen that consists of the entities battery (5 kWH), load, PV system (6 kWp), the grid and the energy management system. As an example, it shall be assumed that the PV system may only feed in 70% of its nominal power. If the power is above this limit, the power is curtailed resulting in financial losses. These specifications correspond to the current regulation by the German EEG 2021 for PV systems up to 25 kWp.

To avoid this effect a corresponding schedule can be calculated that leaves enough free capacity in the ESS to take over the peak power of the PV-system. The corresponding elements that the EMS needs to include can be seen in Figure 1.

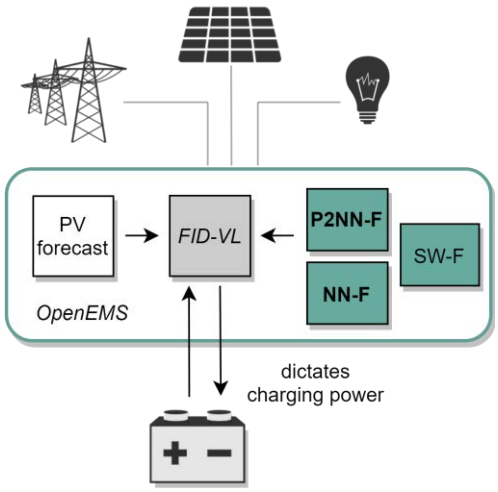

Figure 1: Important elements of the EMS

Central to the setup is the feed-in delimiter FID-VL, which is described in detail in [9]. We use the virtual limit operation strategy to measure the performance of the two proposed forecasting models and the benchmark method.

This strategy requires a forecast of PV production, current measurements of active production and load, the current battery state of charge (SoC) and finally - and most importantly for us - one of the three available load forecasts.

To be able to evaluate each of the proposed models individually too and to relate them towards a lower boundary, we also want to evaluate the feed-in delimiting without forecasts. For this we use the simple feed-in delimiting strategy of FID-VL. Additionally, to relate our findings towards the upper boundary, we perform the simulation with a perfect load forecast. This displays the maximum possible performance.

The electrical connection of the components is depicted in Figure 2. It is also indicated that for the hardware setup of the system not all the components were actually added but that only the battery and the inverter were really installed while the other components were simulated inside the EMS framework.

The EMS that we use to integrate our load forecasting solutions into is Open-EMS. OpenEMS is an open source EMS for PV storage systems in residential buildings written in Java. It supports a great number of hardware components that are relevant for PV storage systems. It features on-board implementations of communication protocols like Modbus and Mbus and is easily expandable due to its flexible and dynamic modular architecture. The architecture and communication in OpenEMS derives from the usage of the OSGi framework, which defines a dynamic component system [10]. This is important for the practical feasibility. Furthermore, it features built-in simulation entities like battery and PV simulators and allows to connect to time-series databases for exterior inputs. For details and documentation of OpenEMS see [11].

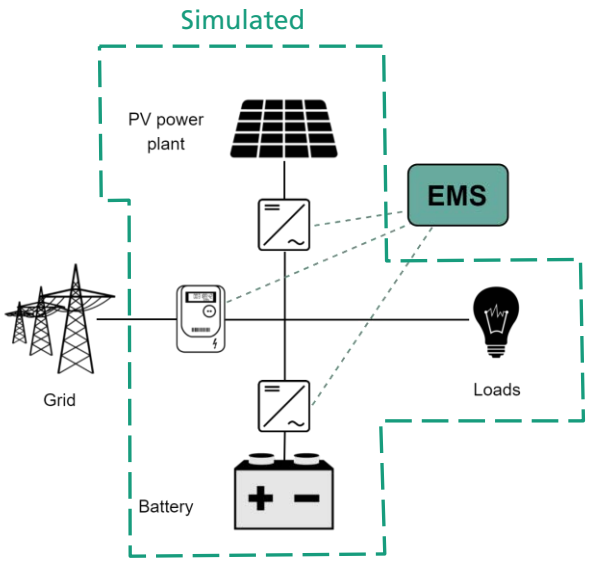

Figure 2: Components of the investigated energy system. The dashed line indicates the components that were simulated.

IV. Results

A. Results for the load prediction algorithms

In the following, we evaluate NN-F and P2NN-F regarding their error minimization performance. We explain the evaluation of the two models separately, as they differ in their prerequisites and concepts.

The evaluation of NN-F is straightforward. We now make use of the testing data that we extracted. We let our model predict loads for all testing samples and measure the respective deviations from the expected (or actual) loads. This requires only the selection of error metrics. We choose MSGE, RMSE and MAE.

For P2NN-F, we must test each household individually. For each of the households, we let the model choose the forecaster on basis of a continuous load series out of the time span of the test data partition. Then, it can output its forecast for all test samples of the test data of that household. We then average the results of each evaluation. In Table 2 we list performance estimations of all three forecasts evaluated on the same test data.

TABLE 2

	MSGE	MSE (kW)	MAE (kW)	MAPE (%)
NN-F	0.558	0.443	0.422	184
P2NN-F	0.582	0.474	0.442	206
SW-F	0.749	0.677	0.533	232

The results indicate that the best performing model is NN-F. Compared to SW-F, NN-F reduces MSGE by 26%, MSE by 35% and MAE by 21% or an averaged total of 111 W. SW-F itself still performs very well for its complexity. That is different with P2NN-F. Although it outperforms SW-F in all considered error metrics too, it is clear that P2NN-F cannot reach the precision of NN-F.

This is surprising given the additional model complexity of P2NN-F. In the design of P2NN-F, we aim for a more precise and individualized forecast of a household. We expect the clustering to hand over to each forecaster only the type of households that it has been trained with. Apparently, the clusters are not dense enough to separate consumer types. The alignment of new households towards one of the 6 clustered profiles cannot sufficiently fulfil the task of profiling. The reason for that could lie in the data pool or the clustering and alignment approach or both. If the data pool does not yield significantly different consumer types, the model cannot work. But it is also possible that the alignment approach of new households is not accurate enough. Analyzing the exact problems and optimizing the model parameters could be an interesting subject for future work.

B. Results for system simulation

The averaged saving per month and household for the three forecasting models and the execution of FID-VL with a perfect forecast are shown in Figure 3. It has to be noted that these findings are not directly transferable to practice. The simulation setup simplifies in using a perfect PV forecast and disregarding electric conversion losses. Also, the monetary evaluation is primarily for an intuitive comparison and is only meaningful in the context of the underlying simulation parameters.

While the perfect forecast is able to save an average of 470 ct, NN-F saves an average of 435 ct - only 7% less. Because of the high amount of energy saved from curtailing, SW-F achieves an averaged saving of 420 ct. Lastly, P2NN-F saves an average of 386 ct.

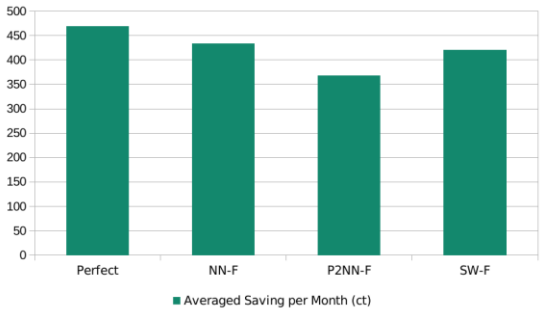

Figure 3: Averaged savings per months for the different load prediction algorithms compared to the perfect prediction

V. Conclusion

In this work we presented two DNN based load forecast concepts of which one performed better than a comparable algorithm that is based on a standard “same day” approach. The load prediction was included into an EMS that controlled a simulated energy system with the aim to reduce curtailment due to a violation of the 70% feed in limit. As EMS platform the open source framework OpenEMS was used. The results of the system simulation also showed a reduction of the overall curtailment losses when the improved load forecaster was used. The OpenEMS implementation developed in the framework of this work can now be easily tested on a real world system.

REFERENCES

[1] UMass Trace Repository. Smart* data set for sustainability. http: //traces.cs.umass.edu/index.php/Smart/Smart. Accessed: 2020-29-01.

[2] Heng Shi, Minghao Xu, and Ran Li. Deep learning for household load forecasting - a novel pooling deep rnn. IEEE TRANSACTIONS ON SMART GRID, VOL. 9, NO. 5, SEPTEMBER, 2018.

[3] Daniel L. Marino, Kasun Amarasinghe, and Milos Manic. Building energy load forecasting using deep neural networks. IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, 2016.

[4] Martin Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorow.org.J.

[5] Fran_cois Chollet et al. Keras. https://keras.io, 2015.

[6] R. Saidur, H.H. Masjuki, M. Hasanuzzaman, and G.S. Kai. Investigation of energy performance and usage behavior of domestic refrigerator freezer using clustering and segmentation. Journal of Applied Sciences 8 (21), 2008.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825-2830, 2011.

[8] Felix Schnorr and Heinrich Hinze. Erstellung von lastprognosen für den elektrischen strombedarf von einfamilienhäusern. https://pvspeicher.htw-berlin.de/veroeffentlichungen/ abschlussarbeiten/, 2014. Accessed: 2020-04-02.

[9] Joseph Bergner, Johannes Weniger, and Tjarko Tjaden. Pvprog-algorithmus. https://pvspeicher.htw-berlin.de/veroeffentlichungen/daten/pvprog/, 2016. Accessed: 2020-23-02.OSGi Alliance. Specifications. https://www.osgi.org/developer/specifications/. Accessed: 2020-21-02.

[11] OpenEMS Association. Documentation of openems. https://openems.io/. Accessed: 2020-04-02.