Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting

Henning Wilms,
Student Member IEEE
Automation of Complex Power Systems
E.ON Energy Research Center
RWTH Aachen University
Aachen, Germany
Email: hwilms@eonerc.rwth-aachen.de

Marco Cupelli,
Senior Member IEEE
Automation of Complex Power Systems
E.ON Energy Research Center
RWTH Aachen University
Aachen, Germany
Email: mcupelli@eonerc.rwth-aachen.de

Antonello Monti,
Senior Member IEEE
Automation of Complex Power Systems
E.ON Energy Research Center
RWTH Aachen University
Aachen, Germany
Email: amonti@eonerc.rwth-aachen.de

***Abstract*—In this paper we propose a *sequence-to-sequence* machine learning architecture for time-series forecasting based on recurrent neural networks. This architecture can be used as a general purpose forecasting method and is evaluated for the application of short-term electric load forecasting in this paper. The proposed sequence-to-sequence architecture[1] combines elements of auto-regressive forecasting techniques with multivariate regression by including exogenous variables for each forecasted time step as well as previous values when inferring forecasts. We assess the proposed architecture on a load data set provided by the Global Energy Forecasting Competition. The conclusion is that it outperforms other machine learning forecasting techniques as well as time-series analysis methods.**

***Index Terms*—Time-Series Forecasting, Load Forecasting, Regression, Auto-Regressive, Recurrent Neural Networks, STLF**

I. Introduction

Neural networks and other machine learning approaches have recently been used for regression tasks, especially *time-series forecasting* with application in the energy sector [1], [2]. Regression as a methodology to forecast loads is a relevant application for the future energy grid [3]. Short-term load forecasting (STLF) considers load forecasts with a time horizon of up to two weeks. Load forecasts with time horizons of less than one day form a subcategory within STLF [4]. These short-term load forecasts are particularly useful for micro-grid operation to optimize the utilization of available resources, prolong islanding mode or reduce carbon footprints. Dispatch scheduling, demand side management and maintenance scheduling in conventional power systems also rely on short-term load forecasts [5].

Forecasting short term electric loads is difficult because of their nonlinear and non-stationary characteristics. As such, electric loads are influenced by temperature, seasonal effects or humidity (amongst others) calling for a multivariate regression approach. However, univariate regressions still leverage some auto-correlation for short-term forecasting but are limited by the nonlinear characteristics of electric load time-series [1], [6]. The proposed forecasting architecture aims at exploiting both univariate and multivariate effects for a better forecasting accuracy.

Time-series contain inherent information in their succession and auto-correlation of values. As such, *auto-regressive* or *univariate* forecasting exploits these time-series dependencies to forecast following values from its previous values utilizing linear relationships between historic and future values. In such auto-regressive forecasting approaches the dependent variable is used to describe the same dependent variable at a different time step. Alternatively, time-series can be forecasted by utilizing supplementary information in the form of independent variables that describe the forecasted value, constituting a *multivariate regression* [7].

One of the several research focuses within the machine learning domain lies on *time series forecasting*. *(Deep) recurrent neural networks* (RNNs) show various advantages and an improved performance when learning time-series dependencies within load forecasting applications to address the above mentioned issues [1], [2], [8], [9].

In this paper we propose a **sequence-to-sequence forecasting architecture** based on recurrent neural networks that combines aspects of univariate and multivariate regression. RNNs exploit multivariate inputs to map these inputs to regress onto desired outputs by finding nonlinear dependencies. At the same time, the architecture is explicitly fed the previous output from the previous time step to add to the auto-recursivity of the forecasting approach. Furthermore, a framework is presented that combines best practice methods for machine learning based forecasting tasks with specific elements of the developed sequence-to-sequence architecture. It provides the training scheme for the architecture as well as data preparation methods required to boost the performance. Finally, the architecture is assessed on the load data set of

[1]The implementation is available at: https://github.com/HenWil13/ieeeINDIN18

DOI: 10.1109/INDIN.2018.8471953.
Publisher version: https://ieeexplore.ieee.org/document/8471953

the Global Energy Forecasting Competition 2014 [10] and the results are compared to various time-series forecasting techniques.

For the context of this paper an *architecture* refers to a general neural network set-up that describes the general workings of the graph. A *model* is a specific instance of an architecture that is fitted, trained and optimized for a specific data set. Exogenous, independent variables provide additional explanatory context to each forecasted value and are also called *features*, whereas the dependent target variables, i.e. the forecast variables are referred to as *labels*.

II. Recurrent neural networks for sequence learning

RNNs possess cyclic connections within their network of neurons that enable them to capture the dynamic behavior of sequences. As the number of cycles of a RNN is arbitrary, the forecast horizon can be chosen and adjusted dynamically. Fig. 1 shows a recurrent neuron with a cyclic connection and shows how this connection is rolled out step by step over a sequence. For time-series forecasting, the temporal dependencies of the time-series can be captured within these cycles. Therefore, the neuron is rolled out further by one step for each forecasted time step, thus producing one output for each time step it is being rolled out over. It is important to note, that the rolled-out neuron is the same neuron for each time step and uses the same trained variables, i.e. *weights* and *biases* for each of these steps to infer the output value.

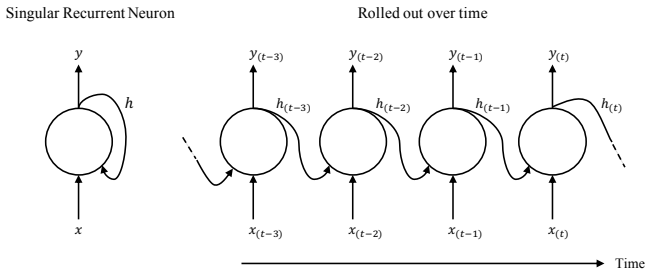

Fig. 1. Recurrent neuron rolled out over time

RNNs store the context of the sequence in a recurrent hidden state h_t, whose activation is dependent on the activation of the previous time step h_{t-1} as well as the current input x_t. For all time steps other than $t=0$ during which the state is initialized to 0, the update of the hidden state follows

$$\boldsymbol{h}_t = \sigma(W\boldsymbol{x}_t + U\boldsymbol{h}_{t-1}) \tag{1}$$

where σ is a non-linear function, called the *activation function* and W and U are weight matrices controlling the recurrent interaction between steps, states and exogenous inputs [9], [11]. As activation function the *tanh-function* has been used in the proposed architecture. Deep RNNs have hidden layers, stacking layers of the depicted RNN neurons above each other.

A. RNN training

We train our RNN models with *backpropagation* using *gradient descent* (GD) on *mini batches* to update the variables within the network [12]. GD calculates the partial derivatives of a loss function for each of the trainable variables within the network. The loss function evaluates the goodness of fit between the network's predictions and the desired labels. Mini batches are used to increase training efficiency, as it is expected that the calculated gradient is about the same for a random batch of the data set as if calculated on the entire data set. The training operation itself then consists of updating all the trainable variables using the calculated partial derivatives with the aim of minimizing the loss function. The update equation for the weights $\mathbf{w}$ is

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathrm{w}} F \tag{2}$$

where η describes the *learning rate* and defines the amount by which the weights are updated along the calculated partial gradients $\nabla_{\mathrm{w}} F$. The partial gradients of the loss function F are calculated with respect to the trainable variables $\mathbf{w}$ for the entire mini batch by GD. This process works analogously for updating the biases of the RNN [9]. However, these gradients often vanish or explode for long sequences or deep RNNs making it hard to train RNNs and extract longer time dependencies [13], [14]. Two approaches provide solutions to this problem and will be utilized within the proposed framework:

- Gradient Clipping and
- RNN Cells.

Gradient clipping limits the norm of the gradients to a threshold [15]. This works especially well to deal with exploding gradients and is used in the proposed sequence-to-sequence model for this purpose. RNN cells deal with exploding or vanishing gradients by using so called recurrent units or *cells*. These cells provide the cyclic self-connection with a value of 1. The information flow is controlled by gates operated to restrict or allow for the state of a cell to be updated. This way the outer gradient remains constant and simply a cell's state is updated with each time step, whenever the gates allow for such an update [11], [15], [16]. Long Short-Term Memory (LSTM) cells [16], Gated Recurrent Units (GRU) [17] and Layer-Normalizing LSTM (LN-LSTM) cells [18] have been evaluated to produce sequence-to-sequence models and are part of the proposed framework. As they all operate in a similar fashion, only the LSTM cell as widely used cell type is described in more detail in the following section.

B. Long short-term memory cells

LSTM cells as depicted in Fig. 2 and described in [11], [16] have three outer connections:

- Output $\boldsymbol{y}$ which is equal to the cell activation $\boldsymbol{h}$,
- Memory cell state $\boldsymbol{s}$ and
- Inputs, i.e. exogenous variables $\boldsymbol{x}$.

Fig. 2 further shows the different gates of the LSTM cell. Each gate is controlled by a fully connected, simple feed forward neural network:

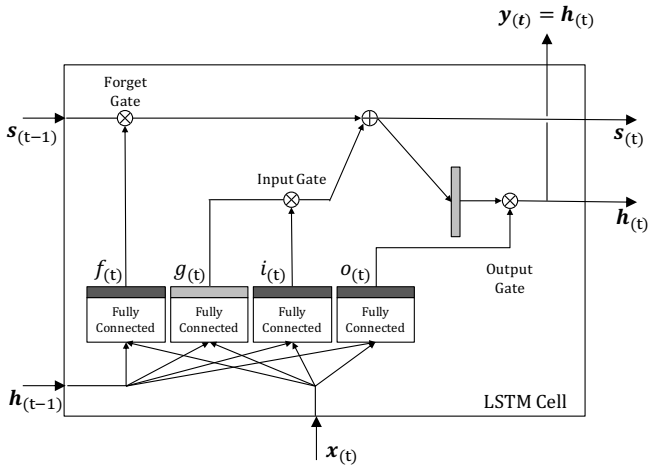

Fig. 2. LSTM Cell

- Forget gate, controlled by $\boldsymbol{f}_{(t)}$
- Input gate, controlled by $\boldsymbol{i}_{(t)}$ selecting what to include from the input node $\boldsymbol{g}_{(t)}$ and the
- output gate, controlled by $\boldsymbol{o}_{(t)}$.

In Fig. 2 the lighter grey boxes represent a *tanh*-activation function ϕ and the darker grey boxes represent a *sigmoid*-activation function σ. The $\oplus$-symbol denotes an element wise addition of the vectors, the $\otimes$-symbol an element wise multiplication. Fig. 2 can be summed up with the following six equations:

$$\boldsymbol{f}_{(t)} = \sigma(W_{f,x}\boldsymbol{x}_{(t)} \oplus W_{f,h}\boldsymbol{h}_{(t-1)}) \oplus b_f \tag{3}$$

$$\boldsymbol{i}_{(t)} = \sigma(W_{i,x}\boldsymbol{x}_{(t)} \oplus W_{i,h}\boldsymbol{h}_{(t-1)}) \oplus b_i \tag{4}$$

$$\boldsymbol{g}_{(t)} = \phi(W_{g,x}\boldsymbol{x}_{(t)} \oplus W_{g,h}\boldsymbol{h}_{(t-1)}) \oplus b_g \tag{5}$$

$$\boldsymbol{o}_{(t)} = \sigma(W_{o,x}\boldsymbol{x}_{(t)} \oplus W_{o,h}\boldsymbol{h}_{(t-1)}) \oplus b_o \tag{6}$$

$$\boldsymbol{s}_{(t)} = \boldsymbol{o}_{(t)} \otimes \boldsymbol{i}_{(t)} \oplus \boldsymbol{s}_{(t-1)} \otimes \boldsymbol{f}_{(t)} \tag{7}$$

$$\boldsymbol{y}_{(t)} = \boldsymbol{h}_{(t)} = \phi(\boldsymbol{s}_{(t)}) \otimes \boldsymbol{o}_{(t)} \tag{8}$$

where W denotes the different weight matrices with the connections of each weight matrix indicated by its indices and b denotes the bias terms of each of the fully connected layers.

GRU cells are a more simplified version of the described LSTM cell as they do not have a separated memory cell state but store their memory in the overall $\boldsymbol{h}$-state. Further, they combine the forget and input gate controller into one and do not need an output gate [17]. The LN-LSTM cell also used in the proposed sequence-to-sequence architecture works very similar to the LSTM cell but adds layer normalization in between each hidden layer to rescale the outputs from the previous layer for the subsequent layer [18].

Considering the STLF use case, the inputs $\boldsymbol{x}$ include the features of the data set and comprise measurements such as temperature and weather conditions, time of the day, day of the week, months or information on holidays. Outputs $\boldsymbol{y}$ would be the load predictions of one or more zones.

Standard RNN architectures using LSTM cells have been used in [1], [2], [8] and [9] and are depicted in Fig. 3. These standard RNNs consist of one or more layers of RNN cells with a cyclic self-connection. RNNs containing more than one layer form a deep network. These RNNs forward their states through their self-connection from one time step to the next time step within each of the layers as the cells are being rolled out over time. Inferred outputs are derived from one or multiple layers for each time step.

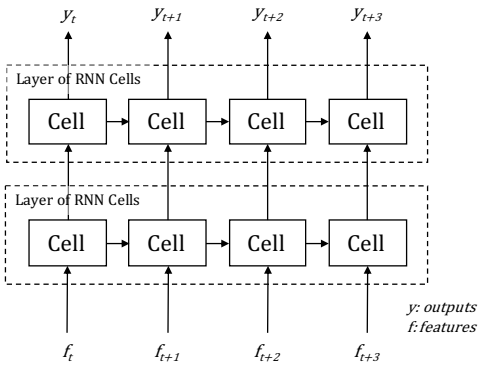

Fig. 3. Standard RNN architecture

Sequence-to-sequence architectures build on these standard RNNs for their basic set up within their encoder and decoder, where the encoder does not produce any outputs and the decoder does not take any exogenous inputs. This RNN based encoder-decoder structure is described in the next section.

III. Sequence-to-Sequence architectures

Sequence-to-sequence architectures have first been introduced in [19] with translation tasks in mind. The general structure of such sequence-to-sequence architectures (depicted in Fig. 4) consists of an *encoder* and a *decoder*.

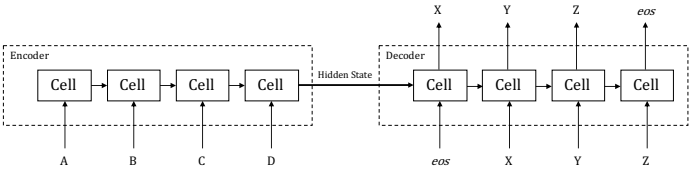

Fig. 4. Sequence-to-sequence architecture as defined in [19]

Here, the encoder is fed an input sequence, i.e. a sentence in the original language. It encodes this sequence within the *hidden state* of the RNN cells where the last RNN cell of the encoder contains a full representation of the input sequence within its hidden state. The decoder takes the full input sequence representation and decodes the hidden state into the target language. It requires an *EOS token* (end of sequence) in the first step of the roll-out to mark the beginning of the target sequence roll-out. From the EOS token onwards, the decoder

unrolls the hidden state in the target language, re-feeding itself the previous output for each time step. At the end of the target sequence it terminates with another EOS token.

A. Proposed architecture for time-series forecasting

The basic idea for our proposed sequence-to-sequence architecture (Fig. 5) combines ideas from

1) standard RNN architectures and
2) sequence-to-sequence architectures.

The standard RNN allows for inclusion of exogenous variables, i.e. features, whereas the decoder part of the sequence-to-sequence architecture enables the self-feeding, auto-regressive element during inference. In this set-up, the encoder "learns" a hidden representation of the historic time-series values up to time step t by feeding the encoder previous values of a feature time-series as well as a *t-1*-shifted label time-series. The final internal representation of the time-series is passed to the decoder as hidden state. This hidden state thus includes a representation of the time-series up to the first forecastable label l_t.

The combination thus consists in building a sequence-to-sequence architecture using the standard RNN setup. However, in comparison to the sequence-to-sequence architecture used for translation tasks, the proposed sequence-to-sequence architecture takes exogenous variables as external inputs in addition to the previous values and thus derives its predictions using both historic, univariate inputs as well as exogenous multivariate features.

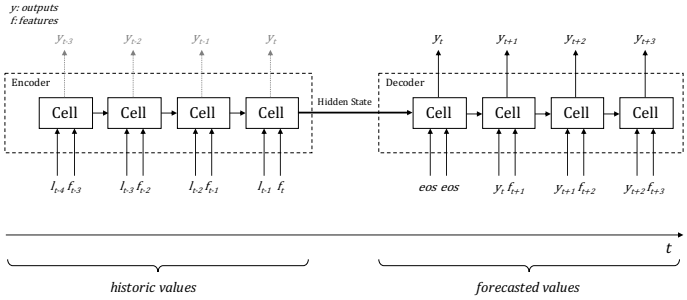

Fig. 5. Proposed sequence-to-sequence architecture for time-series forecasting, greyed out labels of the encoder are omitted during inference and only relevant during training operations

In the proposed architecture, both the encoder and the decoder are fed the same type of inputs: for each time step t the RNN cell takes the features (e.g. temperature measurements) of the same time step t and the labels (observed target values of the load time-series) from the previous time step *t-1*, thus including exogenous variables at each time step for an otherwise auto-regressive forecasting architecture. Encoder and decoder lengths are not fixed and can be chosen and tuned for both best performance during training as well as in accordance with the desired forecast horizon during inference. The proposed architecture does not terminate with an EOS token for the last forecasted step, as there is no natural end to a time-series in comparison to a sentence and thus the decoder terminates with inferring the final label of the predefined forecasting horizon.

B. Encoder

The encoder as such is a standard RNN rolled out over time taking externally fed features of each time step t and labels of time step *t-1* as inputs. The outputs of the encoder are omitted as they do not provide an added benefit to the forecast and would simply result in already known values of historic labels. These outputs however are relevant during the training of a sequence-to-sequence model (see section III-D).

C. Decoder

The decoder in its basic structure also follows that of a simple RNN. However, as it necessitates EOS tokens for the initial step as well as a re-feeding of the outputs during inference, special *helper functions* provide this extra functionality.

EOS tokens need to be concatenated to the start of the input sequence for the decoder during training and inference sessions. Besides, the decoder works analogous to the encoder, taking the same combination of features of time step t and *t-1*-shifted labels as inputs to infer the output at time step t. The training operation fits the RNN so that the combination of inputs predicts the desired target value. During training, the helper does not re-feed the outputs but concatenates the *t-1*-shifted labels to the features of time step t thus simulating a perfectly predicted output at time step *t-1*. This is to ensure that the underlying neural network can learn the auto-regressive element from re-feeding the desired target value that a perfect RNN would re-feed itself. During inference, these labels are yet unknown and the predicted outputs need to be re-fed and joined with the remaining features. Training and inference helper functionalities are depicted in Fig 6.

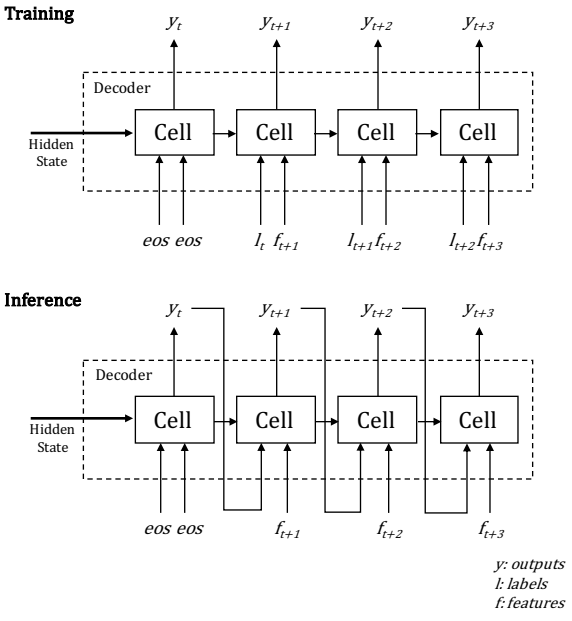

Fig. 6. Label and feature concatenation by the training helper as well as the re-feeding operation of outputs by the inference helpers

D. Overall framework

Fig. 7 shows the overall framework for the sequence-to-sequence architecture and describes the data preparation and training procedure for obtaining a specific sequence-to-sequence model as well as the inference operation.

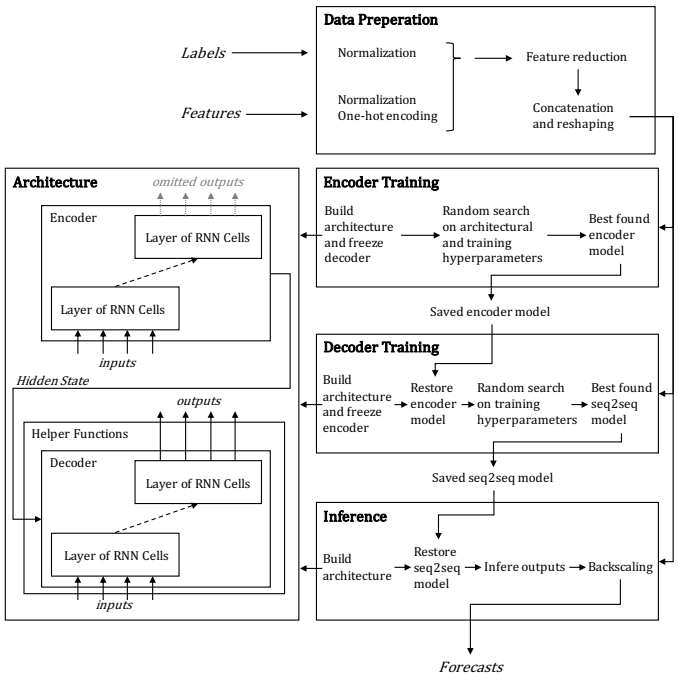

Fig. 7. Sequence-to-sequence (seq2seq) framework for data preparation, training and inference

Data preparation is the first step within the framework. Numerical features and labels need to be rescaled because neural networks in general perform better when all the inputs are from a similar range, as the GD would otherwise be biased towards larger numerical values. For the sequence-to-sequence architecture, features as well as labels must be rescaled, as they jointly form the model inputs. Therefore, the final model outputs need to be scaled backwards for inference. For the used data set (see section IV-B) we have found that standardization yields the best results. Categorical features are one-hot encoded. For the STLF problem at hand hours, days, months and years represent the categorical features. After one-hot encoding, their information is then represented by arrays of ones and zeros.

Generalization is the neural network's capability to learn patterns within the training data set and infer predictions from an unknown data set. *Over-fitting* is the phenomenon when the neural network starts learning the training data set "by heart" and stops learning general patterns thus losing its generalization capabilities. This happens especially in deep neural networks with many units and degrees of freedom. We use *Dropout* as a means of *regularization* to reduce the overfitting of our sequence-to-sequence models. During the training session the neural network's degrees of freedom are impeded to force the neural network to focus on the underlying patterns of the data set and ensure generalization capabilities [20], [21].

Parameters such as the number of hidden layers, number of units, regularization or the learning rate are also called *hyperparameters*. They specify the overall set-up of an architecture that is to be used for a specific model. As it is not possible to define the best combination of hyperparameters beforehand, *random search* trains several different versions of the architecture with different random hyperparameter combinations of a predefined hyperparameter search space and evaluates these combinations. This does not lead to the overall best possible combination of hyperparameters but results in a close fit, depending on the number of randomly chosen hyperparameter combinations and the predefined search space [21].

The training adjusts the trainable variables of the network so that the desired target value can be inferred from the provided inputs (see section II-A). For the proposed sequence-to-sequence architecture there is a two-fold training operation, performed one after the other:

1) encoder training and
2) decoder training.

Encoder training is performed before decoder training to ensure that the best possible representation of the historic time-series is embedded in the hidden state that is passed to the decoder making it easier for the decoder to learn from this starting point. During encoder training, only the trainable variables of the encoder are adjusted and the loss function is calculated using only the omitted outputs. After the encoder is fully trained, the decoder is trained using only the decoder variables and the decoder outputs, i.e. the models final inferred predictions are used for the loss calculation.

Random search is applied to find the best hyperparameter combination for each specific data set (see section II-A). There are two different kinds of hyperparameters in the case of our sequence-to-sequence architecture:

- Architectural hyperparameters and
- Training hyperparameters.

Architectural hyperparameters need to be the same for both the encoder as well as the decoder and are hence only part of the encoder random search. The best found architectural hyperparameter combination for the encoder must then be the same for the decoder. Training hyperparameters can be different for encoder and decoder. Table I provides an overview over the two hyperparameter classes.

IV. Evaluation

A. Benchmark forecasting methods

We compare our proposed sequence-to-sequence architecture to two machine learning approaches as well as to two methods from the time-series analysis (TSA) domain. From the machine learning domain, we use a **standard RNN** following

TABLE I
OVERVIEW OVER ARCHITECTURAL AND TRAINING HYPERPARAMETERS

Architectural Hyperparameters	Training Hyperparameters
Number Layers	Learning Rate
Number Units	Gradient Clipping Value
Cell Type	Batch Size
	Dropout Keep Probabilities

the established best practice approach described in [8] as well as a **random forest regressor** implemented following the description of [22]. Further, we performed TSA-based forecasting using **second degree exponential smoothing** and **auto-regressive integrated moving average** (ARIMA). For the TSA forecasting we used the default off the shelf implementations within pandas for exp. smoothing and statsmodels for ARIMA.

B. Test data set

As test data set we used the load data set of the Global Energy Forecasting Competition 2014 [10]. It contains hourly load measurements as target time-series as well as 25 supplementary temperature readings as well as time stamps (string containing the date and time information). In total, the data set has a length of 60,600 entries. The first 80% of the data set are used for training and the remaining part for validation and evaluation. The minimal value l_{min} of the load time-series is 16.1, the maximum value l_{max} is 317.5.
The test data set shows a cyclic auto-correlation that is especially strong for a lag of under 500 as shown in Fig. 8. The auto-correlation in the data set is expected to benefit the auto-regressive element within the sequence-to-sequence architecture as well as the benchmarks performing TSA based forecasts.

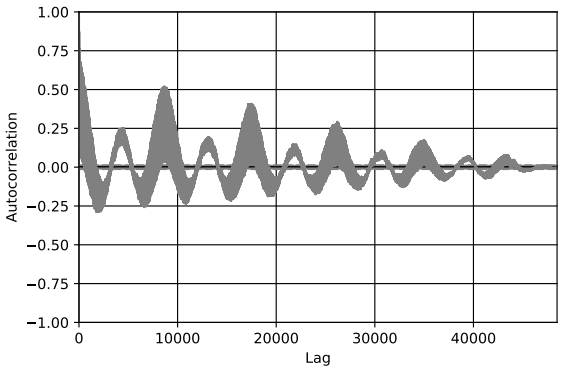

Fig. 8. Auto-correlation plot of the test data set

C. Testing Methodology

A forecast horizon of 12 hours is used to evaluate the proposed forecasting architecture. All the used benchmark methods have been set up, trained or optimized using their respective best practice approaches. The benchmarks also perform a 12 hour forecast for comparison. The validation data set is used for the evaluation of the benchmark methods as well as our proposed forecasting architecture. It is reshaped into sequences of 12 values. Each method forecasts all the sequences within the validation data set and the evaluation metrics are calculated on each of the resulting sequences and are finally reduced to their mean value for the overall comparison.
Three evaluation metrics were chosen for the evaluation. The R^2*-accuracy measure* was used although being uncommon because it provides a not skewed, scale independent interpretation of the goodness of fit for each of the models. It describes the explained variance by the model and input data as part of the variance within the underlying targets. A perfect fit results in a R^2 of one [23], [24].
For better interpretation of the forecasting accuracy the *root mean square error* (RMSE) is also calculated:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - l_i)^2} \tag{9}$$

with *i* denoting each output y_i and label l_i for all the sequences of the validation data set comprising n instances, i.e. overall number of hourly values.
Furthermore, the *normalized RMSE* (nRMSE$_{\%}$) is calculated to add interpretability by dividing the RMSE by the overall labels' scale $l_{max} - l_{min}$ (see section IV-B):

$$nRMSE_{\%} = \frac{RMSE}{l_{max} - l_{min}} \times 100 \tag{10}$$

D. Results

Table II shows the results for the different forecasting methods and their respective R^2, RMSE and nRMSE$_{\%}$ values.

TABLE II
RESULTS OVERVIEW AND COMPARISON

Model	R^2	RMSE	nRMSE$_{\%}$
Sequence-to-sequence RNN	0.9497	10.82	3.59
Standard RNN	0.9168	14.37	4.77
Random Forests	0.8335	18.31	6.07
ARIMA	0.7553	18.72	6.21
Exponential Smoothing	0.6245	24.12	8.00

The results clearly show that the proposed sequence-to-sequence architecture outperforms other state of the art machine learning approaches on the GEFCom data set. Compared to TSA, the superiority of the sequence-to-sequence architecture is clearly visible as well.

V. DISCUSSION AND CONCLUSIONS

There are three particularities that need to be evaluated in future research:

1) effect of encoder accuracy,
2) effect of decoder inaccuracy on the forecast deterioration over the forecasting horizon and
3) performance on data sets with less auto-correlation.

During the training of our sequence-to-sequence architecture, we noticed that the overall performance of the decoder is strongly dependent on the accuracy of the encoder, i.e. the accuracy of its encoded hidden state that provides the first step of the decoder. This effect should be evaluated further to determine if the sequence-to-sequence architecture is outperformed by the benchmark methods when the encoder cannot be fitted well enough and wether this threshold can be defined more clearly.
Secondly, we suspect that the decoder's inaccuracy will increase with longer forecast horizons, as the error of each forecasted step will propagate into the following steps due to its re-feeding of the forecasted value. This should also be evaluated further as the benchmark methods could outperform the sequence-to-sequence architecture on longer forecast horizons.
Furthermore, it is interesting to evaluate wether the sequence-to-sequence architecture can still make use of the auto-recursive element when the data set does not show such a strong short-term auto-correlation and continues outperforming the multivariate regressions techniques.

In this paper we have introduced an adaptation of RNN based sequence-to-sequence architectures for time-series forecasting of electrical loads within power systems. The architecture combines univariate and multivariate forecasting techniques. We were able to show that this combination yields better results than strictly univariate or multivariate forecasting methods for a load data set that shows an auto-correlation over the first few lags. This auto-correlation is captured by the auto-regressive re-feeding of the proposed architecture, whereas the non-stationary characteristics are captured by the multivariate element. As many load data set show such a short-term auto-correlation combined with nonseasonal and non-stationary characteristics, the sequence-to-sequence architecture should be considered as an alternative for STLF.
During the random search for the tuning of the hyperparameters, we have found that 2-4 hidden layers with 10-40 cells are sufficient to capture the GEFCom Load data set's patterns and produce reasonable predictions. In general, hidden layers of 3 or more needed high amounts of regularization to prevent overfitting and we deduce that 2-3 layers of 10-20 cells provide sufficient degrees of freedom for the STLF example case at hand. The LN-LSTM cell usually converged quicker than the others when comparing training iterations but took longer in total due to their increased computational complexity.

ACKNOWLEDGMENT

We thank the European Commission for their funding of the InterFlex H2020 project (grant agreement no 731289).

REFERENCES

[1] Z. Jian, X. Cencen, Z. Ziang, and L. Xiaohua, "Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, Conference Proceedings, pp. 1–6.

[2] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, "An overview and comparative analysis of recurrent neural networks for short term load forecasting," *arXiv preprint arXiv:1705.04378*, 2017.

[3] K. D. Orwig, M. L. Ahlstrom, V. Banunarayanan, J. Sharp, J. M. Wilczak, J. Freedman, S. E. Haupt, J. Cline, O. Bartholomy, H. F. Hamann, B. M. Hodge, C. Finley, D. Nakafuji, J. L. Peterson, D. Maggio, and M. Marquis, "Recent trends in variable generation forecasting and its value to the power system," *IEEE Transactions on Sustainable Energy*, vol. 6, no. 3, pp. 924–933, 2015.

[4] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *International Journal of Forecasting*, vol. 32, no. 3, pp. 914–938, 2016.

[5] H. Chitsaz, H. Shaker, H. Zareipour, D. Wood, and N. Amjady, "Short-term electricity load forecasting of buildings in microgrids," *Energy and Buildings*, vol. 99, pp. 50–60, 2015.

[6] D. Xishuang, Q. Lijun, and H. Lei, "Short-term load forecasting in smart grid: A combined cnn and k-means clustering approach," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Conference Proceedings, pp. 119–125.

[7] J. S. Armstrong, *Principles of forecasting: a handbook for researchers and practitioners*. Springer Science and Business Media, 2001, vol. 30.

[8] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE Transactions on Smart Grid*, vol. PP, no. 99, pp. 1–1, 2017.

[9] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[10] T. Hong, P. Pinson, S. Fan, H. Zareipour, A. Troccoli, and R. J. Hyndman, "Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond," *International Journal of Forecasting*, vol. 32, no. 3, pp. 896–913, 2016.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv e-prints*, vol. 1412, p. arXiv:1412.3555, 2014.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Report, 1985.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, Conference Proceedings, pp. 1310–1318.

[16] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, Conference Proceedings, pp. 850–855 vol.2.

[17] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *ArXiv e-prints*, vol. 1406, p. arXiv:1406.1078, 2014.

[18] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv e-prints*, vol. 1607, p. arXiv:1607.06450, 2016. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160706450L

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, Conference Proceedings, pp. 3104–3112.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[22] A. Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[23] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[24] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.