**State of Health Estimation and Prediction of Fuel Cell Stacks in Backup Power Systems**

S. H. Sønderskov, J. W. Ilsøe, D. Rasmussen, D. Blom-Hansen, and S. Munk-Nielsen

This is a preprint of a paper submitted for publication in IEEE Transactions on Industrial Electronics.

# State of Health Estimation and Prediction of Fuel Cell Stacks in Backup Power Systems

***Abstract*—Fuel cell based backup systems are used as telecommunication power supplies, where availability of power is crucial for a reliable service. State of health (SOH) is a useful metric in aiding predictive maintenance actions, that aim to reduce the system down time as well as the operating cost. In this paper, voltage, current, and temperature data from numerous stacks installed in the field to estimate an SOH metric. A long short term memory (LSTM) recurrent neural network (RNN) is trained to predict SOH values six months into the future. Finally, the RNN performance is evaluated on prediction horizons of six months, as well as longer horizons of twelve, eighteen, and twenty-four months.**

***Index Terms*—Fuel cells, Recurrent neural networks, Uninterruptible power systems**

## I. Introduction

FUEL cell based backup systems has become a popular solution in powering telecommunication sites. The fuel cell solution provides a less polluting and less noisy alternative to conventional diesel generators. They are more sustainable for longer runtimes and have easier extendable capacity compared to pure battery solutions. [1]

In backup systems, the availability of power is of paramount importance. Therefore, the system performance should be ensured through appropriate maintenance strategies, monitoring of system parameters, and other measures. Assessing the performance level of a complex system is not trivial and many approaches are described in literature. A popular metric for assessing stack performance is the stack voltage [2], [3]. However, the voltage does not only change with the degradation of the stack, but also with varying operating conditions, such as load and temperature. Therefore, this approach is often used in applications with constant load. Another popular method, especially used for diagnostics and characterization, is based on electrochemical impedance spectroscopy (EIS), which has proven very effective [4]–[6]. However, it requires perturbation of the stack current at a specific operating point and additional hardware for measuring the response.

An alternative method is to estimate a state of health (SOH) metric, i.e. a single number that indicates the condition of the stack. There are no established standards in the way that SoH is derived for a fuel cell. But a number of methods is presented in literature, including voltage based (as mentioned previously), and internal resistance. [7], [8]

In this paper, voltage, current, and temperature measurements from numerous fuel cell stacks installed in the field, are used to estimate the state of health of the stacks. A recurrent neural network (RNN) is trained on the data to predict the state of heath metric six months into the future. The main contributions of this work is a method for estimating SOH of fuel cell stacks in backup systems, where operation is sporadic. As well as a method for predicting future SOH values of individual stacks, based on known SOH metrics from a fleet of stacks installed in the field, which to the authors' knowledge, has not been seen in the literature before.

## II. Data Foundation

Data from 74 stacks is used for the studies in this paper. The stacks are used in telecommunication uninterruptible power supplies, as described in [9]. These systems experience some unique operating conditions, involving long periods of inactive/standby modes and few, sporadic backup scenarios.

To ensure system functionality when backup is needed, the fuel cell stacks are regularly conditioned in so called self-test, where the stacks are powered up to deliver energy to a dummy load, thereby exercising the fuel cells. These self-tests are more regular and consistent, than the backup power scenarios, and are therefore more suited to study the fuel cell degradation over time.

The data, which is used in this work is voltage, current, and temperature of the fuel cell during the before mentioned self-tests. An example of such data from a single self-test of a single stack is shown in Fig. 1. To simplify the dataset and to allow for easier analysis of performance levels, each self-test is reduced to a single data-point - the steady state value - for each of the three variables.

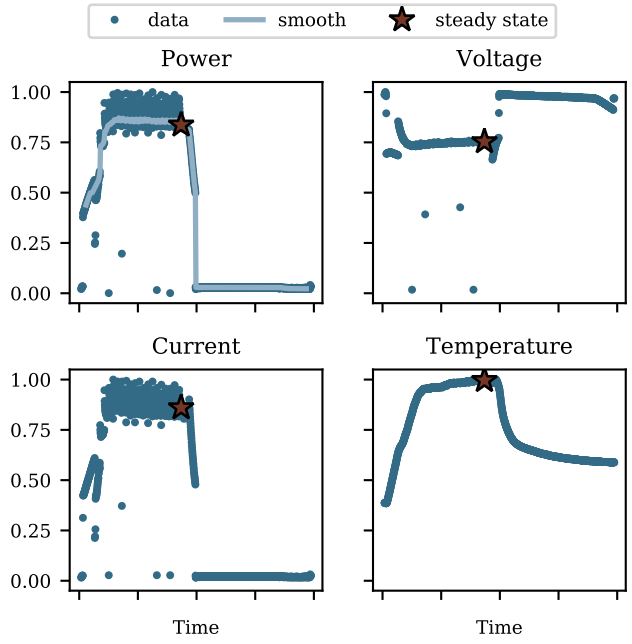


Fig. 1. Self-test example showing the steady state extraction principle.

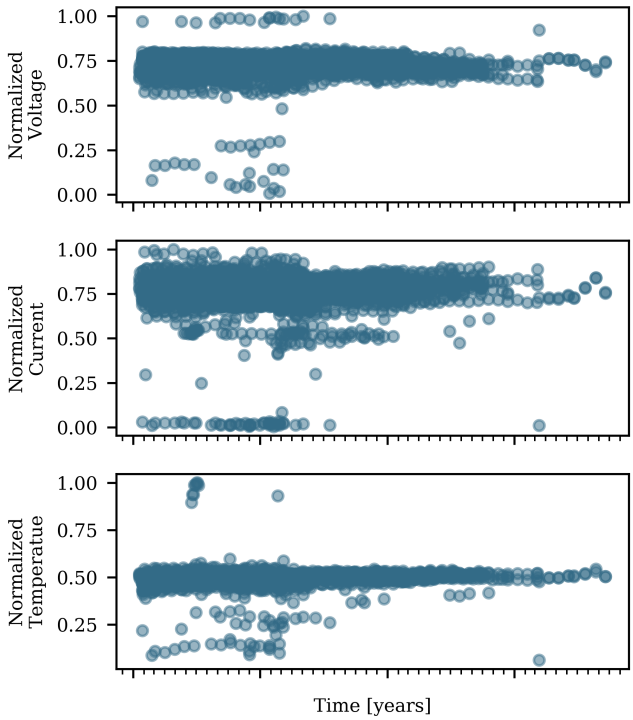


Fig. 2. Steady state voltage, current, and temperature data for all the stacks in the dataset.

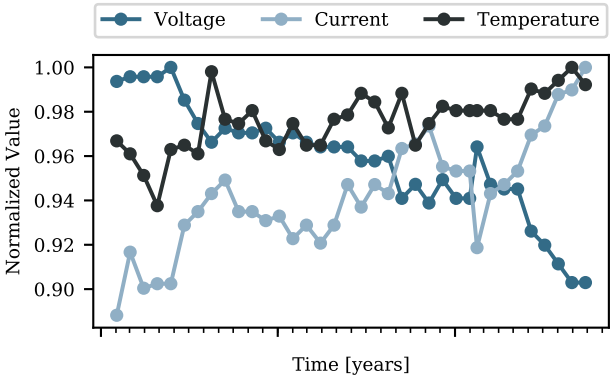


Fig. 3. Steady state voltage, current, temperature evolution of an example stack.

The steady state values are extracted by finding the end of the constant power segment of the data and then taking the variable values at this time-point. The steady state extraction process is described in detail in [10].

Finding these steady state values for each self-test of each stack in the dataset gives the cloud of data, shown in Fig. 2. The figure does not depict to which stack each data point belongs, but merely illustrates the dataspace. An example of how the voltage, current and temperature steady state values evolve in time for a single stack is shown Fig. 3.

## III. State of Health Estimation

To estimate the state of health of any given stack, two reference points are needed; a beginning of life point (BOL; when the stacks is performing as a stack is expected to perform at the beginning of its lifetime), and an end of life point (EOL; when the stack performance has fallen to an unacceptable level).

### *A. Beginning of Life and End of Life Definitions*

The fuel cell manufacturer has provided their definitions of the beginning of life and end of life criteria in the datasheet of the fuel cell. These boundaries are given as polarization curves, i.e. voltage-current curves of the fuel cell, as shown in Fig. 4. The polarization curves are obtained through a specific sequence of current steps in a lab environment with controlled fuel cell temperatures. Throughout the test, the temperature of the fuel cell is kept close to the optimum temperature for that specific current level, i.e. the temperature is lower in the low current steps than at the high current steps.

To allow for comparing values to the polarization curves at any current level, the formula in (1) [11] is fitted to the test data. The formula represents the fuel

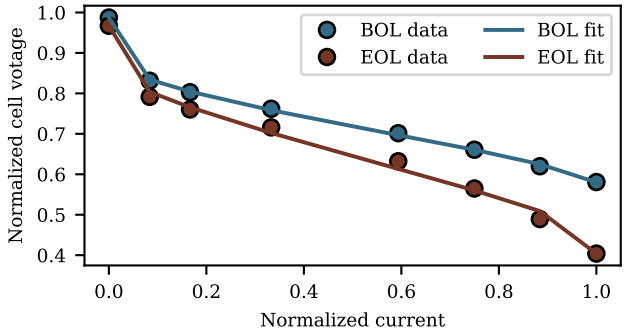


Fig. 4. Beginning of life (BOL) and end of life (EOL) points as defined in the fuel cell manual and fitted curves.

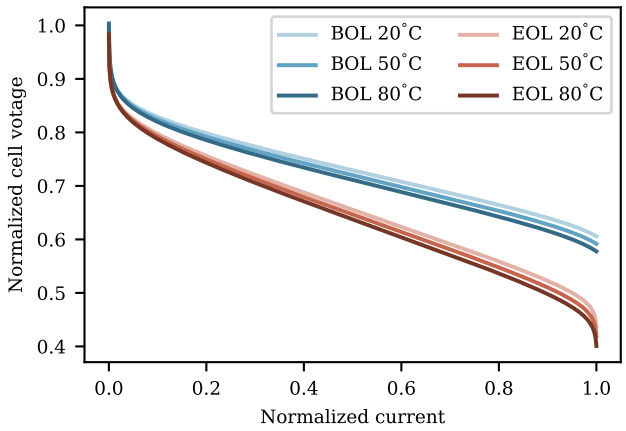


Fig. 5. Fitted polarization curves for beginning of life and end of life criteria at different temperatures.

cell voltage as the reversible open circuit voltage minus the different types of voltage loss in the cell.

$$V_\mathrm{c} = V_\mathrm{r} - (I + I_\mathrm{n})R - \frac{\mathcal{R}T}{2\alpha\mathcal{F}} \ln\left(\frac{I + I_\mathrm{n}}{I_\mathrm{o}}\right) + \frac{\mathcal{R}T}{2\mathcal{F}} \ln\left(1 - \frac{I + I_\mathrm{n}}{I_\mathrm{L}}\right) \quad (1)$$

where $I$ is the cell current, $T$ is the temperature, $\mathcal{R}$ is the universal gas constant ($\approx 8.314\,\mathrm{J\,mol^{-1}\,K^{-1}}$), and $\mathcal{F}$ is the Faraday constant ($\approx 96\,485.332\,\mathrm{C\,mol^{-1}}$). The remaining parameters are the fitting parameters and are described in Table I. The obtained fit is shown for three temperature levels along the datasheet BOL and EOL definitions in Fig. 5. The graphs show that the cell voltage falls slightly for an increased temperature. This is not usually the case for fuel cells and is mainly due to the fact that the parameters of (1), such as $i_o$ and $r$, have temperature dependencies, which are

TABLE I
FUEL CELL POLARIZATION CURVE FITTING PARAMETERS.

| Symbol | Quantity | Unit |
|---|---|---|
| $V_\mathrm{r}$ | Reversible open circuit voltage | V |
| $I_\mathrm{n}$ | Sum current equivalent of fuel crossover and the internal current | A |
| $I_\mathrm{o}$ | Exchange current | A |
| $I_\mathrm{L}$ | Limiting current | A |
| $R$ | Resistance | Ω |
| $\alpha$ | Charge transfer coefficient | - |

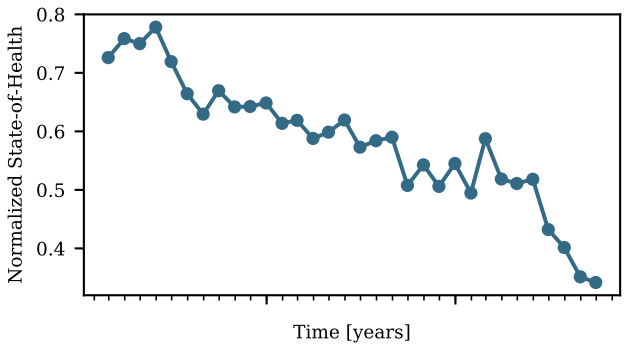


Fig. 6. State-of-Health estimates of an example stack.

not modeled here. However, the variation in voltage caused by temperature is relatively small, and the variation in temperature is limited (between 40°C and 60°C for the vast majority of the data points). Therefore, the presented fit is considered applicable for this specific case.

### B. State of Health Estimation

After obtaining an expression for the cell voltage as a function of current and temperature at the beginning of life ($V_\mathrm{c,bol}(I,T)$), and an expression at the end of life ($V_\mathrm{c,eol}(I,T)$), the state of health of the fuel cell stack is estimated using the formula:

$$SOH(V_\mathrm{c}, I, T) = \frac{V_\mathrm{c} - V_\mathrm{c,eol}(I,T)}{V_\mathrm{c,bol}(I,T) - V_\mathrm{c,eol}(I,T)} \quad (2)$$

where $V_\mathrm{c}$, $I$, and $T$ are the self-test steady state data points of voltage, current, and temperature, respectively. The resulting state of health for an example stack is shown in Fig. 6.

## IV. Recurrent Neural Networks

Recurrent neural networks (RNN) are a variation of artificial neural networks (ANN). The underlying equation in ANN is shown in (3), where $y$ represents the output of a neuron and $x$ is the input, $W$ is a

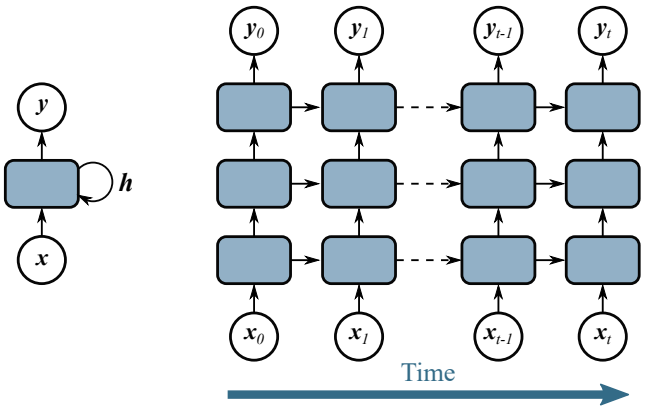


Fig. 7. Left: single-layer recurrent neural network. Right: three-layer recurrent network unrolled through time. The output and possibly some hidden states are used as additional inputs in the next step of the sequential network.

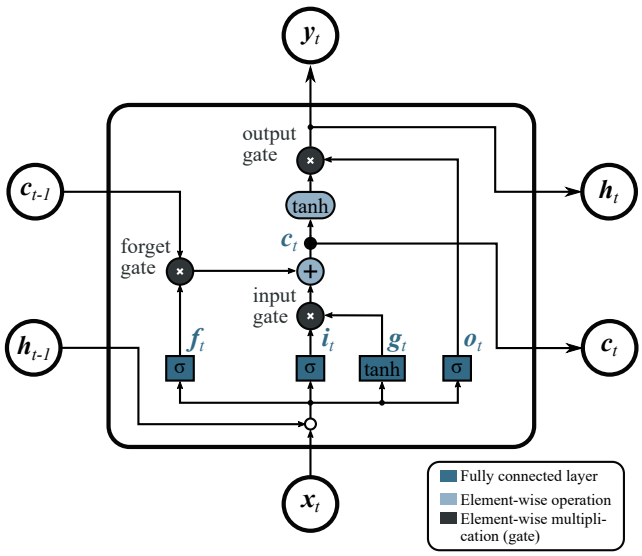


Fig. 8. Long short term memory (LSTM) cell.

weight, and $b$ is a bias. $\sigma$ is an activation function, which adds nonlinearity to the equation. Popular activation functions include the hyperbolic tangent, Sigmoid, and rectified linear unit (ReLU).

$$y = \sigma\left(Wx + b\right) \tag{3}$$

Recurrent neural networks can be viewed as a repetition of multiple ANNs, where the outputs of the ANN layers are used as additional inputs to the following ANN layers. This is represented in Fig. 7 and gives the model a temporal dimension, which makes this type of neural networks ideal for modeling sequential data, such as time-series. In the right hand side of Fig. 7, a representation of an RNN with three layers is depicted as it is unrolled through time. So that, $\boldsymbol{x}_0$ is the input vector at $t = 0$ and $\boldsymbol{y}_0$ is the resulting output when the input is fed through the three layers. For the following time step, not only is $\boldsymbol{x}_1$ input to the first layer, but also the output of the first layer in the previous time step. This use of previous outputs and hidden states can be considered the models memory.

One shortcoming of the generic RNN as the one discussed above, is that in each time step the memory term is multiplied by the same weight, leading to the vanishing gradient problem, where the memory becomes less significant over time, resulting in improper training of the network. However, this problem is solved in variations of the RNN, such as long short term memory (LSTM) RNN networks [12].

### A. Long Short Term Memory Cell

The long short term memory (LSTM) cell is a variation of the recurrent neural network. The LSTM cell has internal layers, referred to as gates, which allow the network to choose what should be remembered and what should be forgotten. A graphical representation of an LSTM cell is depicted in Fig. 8. Additional to using the output of the previous layer in the input of the subsequent layer as in the basic RNN, an LSTM network also uses an internal cell state in the input of the following layer. The process and the mathematics of the LSTM cell is described in the following.

The LSTM network has three internal gates: the input gate, the forget gate, and the output gate. The function of the gates, is to select which parts of the input should be passed through to the next layer. The forget gate selects which part of the cell state from the previous step ($\boldsymbol{c}_{t-1}$) should be passed to the current cell state, and hence which part should be forgotten. The activation vector of the forget gate ($\boldsymbol{f}_t$) is calculated using artificial neuron function on the input ($\boldsymbol{x}_t$) and the output of the previous step ($\boldsymbol{h}_{t-1}$) as shown in (4).

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{\mathrm{f}}\left[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right] + \boldsymbol{b}_{\mathrm{f}}\right) \tag{4}$$

where $\boldsymbol{W}_f$ is the weight matrix, $\boldsymbol{b}_f$ is the bias, and $\sigma$ is the activation function, in this case the sigmoid function.

The input gate has two inputs: the new candidate cell state ($\boldsymbol{g}_t$), and the input activation vector ($\boldsymbol{i}_t$).

The input gate activation function is calculated similarly to the forget gate activation function:

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_{\mathrm{i}}\left[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right] + \boldsymbol{b}_{\mathrm{i}}\right) \tag{5}$$

The candidate cell state is also calculated using the artificial neuron function, only the sigmoid activation function is replaced with an hyperbolic tangent (tanh) activation function:

$$\boldsymbol{g}_t = \tanh\left(\boldsymbol{W}_{\mathrm{g}}\left[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right] + \boldsymbol{b}_{\mathrm{g}}\right) \tag{6}$$

The actual cell state ($\boldsymbol{c}_t$) is obtained by adding the outputs of the input gate and the forget gate, i.e. the cell state is a combination of a selection of the previous cell state data and a selection of data from the input and foregoing output:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \otimes c_{t-1} + \boldsymbol{i}_t \otimes \boldsymbol{g}_t \tag{7}$$

The final gate is the output gate, which determines which parts of the cell state should be output and fed to the next layer and/or next time step. The activation vector of the output gate ($\boldsymbol{o}_t$) is calculated by (8).

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{\mathrm{o}}\left[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right] + \boldsymbol{b}_{\mathrm{o}}\right) \tag{8}$$

Finally, the output ($\boldsymbol{y}_t$ or $\boldsymbol{h}_t$) is calculated in the output gate:

$$\boldsymbol{y}_t = \boldsymbol{h}_t = \boldsymbol{o}_t \otimes \tanh\left(\boldsymbol{c}_t\right) \tag{9}$$

The output is then used for two purposes: 1) as a final output of the RNN or as an input to a following layer in the model (called $\boldsymbol{y}_t$) and 2) as a hidden memory state to the following time step (called $\boldsymbol{h}_t$).

## V. Constructing and Training the Model

The LSTM RNN model is constructed using the Python library Keras [13], which is a high-level API for neural network implementation. The network architecture is chosen to consist of three main LSTM layers with 64, 128, and 64 neurons, respectively. Each LSTM layer is put in a bidirectional wrapper, which effectively makes two LSTM layers: One is trained in normal time order, i.e. from past to future; the other is trained in the reverse time order, i.e. from future to past. This allows the model to use both past and future information during the training. This also doubles the size of the layer, so that a bidirectional LSTM layer, with each 64 neurons will amount to a total of 128 neurons. After the two first LSTM layers, a dropout layer is added. The dropout layer randomly zeros a fraction of the input units, thereby ignoring these inputs in the training process. This helps prevent overfitting the model. Finally a standard fully-connected (dense) neural network layer with six neurons is used for the output layer. The output of neuron 1 in the output layer represents the future value one time step forward, neuron 2 is the value two time steps in the future and so forth.

The implementation of these layers using Keras in Python is shown in Listing 1. And the constructed model is summarized in Table II, showing that more than 460,000 parameters (weights and biases) needs to be estimated during the training process. The output shape column shows the shape of the output of the specific layer in the format ($batch\_size$, $samples$, $units$). The $batch\_size$ refers to the number of stacks that is passed through the network at a time, $samples$ refers to the number of samples per stack, and $units$ is the number of units/neurons in the layer. The First two dimensions are listed as 'None', because these dimensions are of variable size, i.e. different stacks have different number of samples and the number of stacks per batch varies, as explained in Section V-A. The output shape of the output layer (Dense_1), only has two dimensions, namely ($batch\_size$, $n\_pred$), i.e. $n\_pred$ predictions are made for $batch\_size$ stacks. $n\_pred$ is determined by the number of units in the output layer.

Listing 1. RNN model implementation

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout,
    Bidirectional

# Initiate a sequential model
model = Sequential()
# Add first LSTM layer and dropout
model.add(Bidirectional(
  LSTM(64, return_sequences=True),
  input_shape=(None, 1)))
model.add(Dropout(0.2))
# Add second LSTM layer and dropout
model.add(Bidirectional(
  LSTM(128, activation='relu',
  return_sequences=True)))
model.add(Dropout(0.2))
# Add third LSTM layer
model.add(Bidirectional(
  LSTM(64, activation='relu')))
# Add output layer
model.add(Dense(6))
```

### A. Preprocessing

Before training the model, the dataset is split in three groups: train, validate, and test. The training data is used for training the model, i.e. tuning the

TABLE II
RECURRENT NEURAL NETWORK MODEL SUMMARY.

| Layer (type) | Output Shape | Parameters |
|---|---|---|
| LSTM_1 (Bidirectional LSTM) | (None, None, 128) | 33 792 |
| Dropout_1 (Dropout) | (None, None, 128) | 0 |
| LSTM_2 (Bidirectional LSTM) | (None, None, 256) | 263 168 |
| Dropout_2 (Dropout) | (None, None, 256) | 0 |
| LSTM_3 (Bidirectional LSTM) | (None, 128) | 164 352 |
| Dense_1 (Dense) | (None, 6) | 774 |
| Total parameters: | | 462 086 |

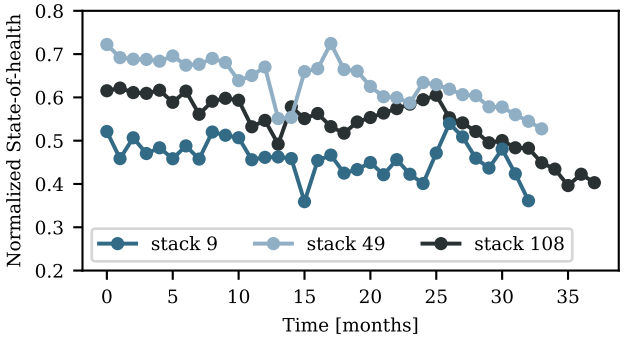


Fig. 9. Testing data; three randomly selected stacks out of the total set of 74 stacks.

weights and biases of the network so that outputs of the network match the expected based on the training input. The validation data is used to valuate the training process and is not used to update the network parameters. The testing data is kept completely separate from the training process. The training data sample contains 56 stacks (84%), the validation sample contains 9 stacks (12%) and the testing sample is 3 stacks (4%).

Secondly, the training data should be grouped in batches, depending on the number of samples per stack. The number of samples for each stack varies between 5 and 39, with most stacks (10) having 15 samples. Consequently, 24 batches are created, containing the data of between 1 and 10 stacks, each.

The final step before actually training the model is to normalize the data. For this purpose, the MinMaxScaler of the Scikit Learn package [14] in Python is used, which scales the data to fit between two defined values, in this case, 0 and 1. The normalized data of the three testing stacks is depicted in Fig. 9.

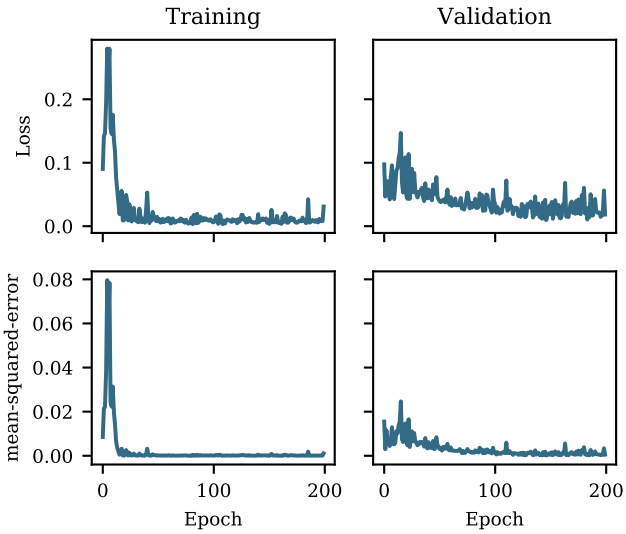


Fig. 10. Training (left) and validation (right) accuracy (top) and loss (bottom) evolution during the training epochs.

### B. Training

The last six samples of each stack is used as labels ($\boldsymbol{y}$), i.e. the reference for the prediction, and the remaining samples are used as inputs ($\boldsymbol{X}$):

```
X = batch[:, :-6, :]
y = batch[:, -6:, :]
```

where `batch` is a batch of stacks with the dimensions ($batch\_size$, $samples$, $features$).

The network is trained on the training data for 200 epochs, using the Adam optimizer [15] and mean absolute error (MAE) as the loss function. After each training epoch, the model is used to make predictions on the validations data. The result of the loss function (MAE) for both training and validation data after each epoch is depicted in Fig. 10 along with the mean-squared-errors (MSE). After 200 epochs, the training loss has dropped to approximately 0.008 and the validation loss to approximately 0.018. The MSE is used as a secondary metric for the accuracy of the training, which is independent from the MAE used for optimization. After 200 epochs the training MSE is 0.0008 and the validation MSE is 0.0005.

## VI. MODEL EVALUATION

The trained RNN is used to predict the last six samples of each of the validation stacks and the result is shown in Fig. 11. Points at and below zero months (on the x-axis) represent the data used for input to the model, i.e. considered past data. Points

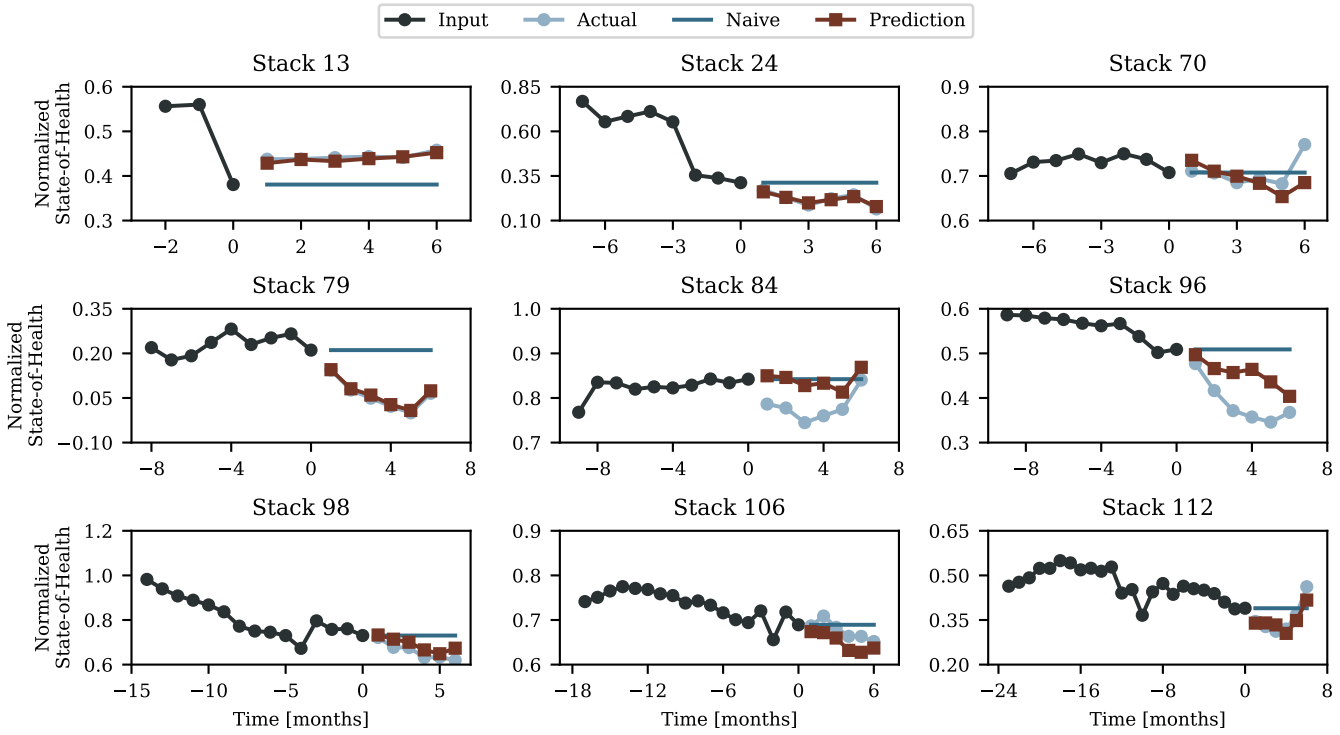


Fig. 11. Prediction results on the validation data. The x-axis zero-point denotes the time of the prediction, such that any negative x-values are past values and are used as input for the recurrent neural network, and positive x-values are future values, which should be predicted by the outputs of the network. The graph shows the input data, the actual future values, the naive predictions, and the RNN predictions.

above zero months are considered future values, either true values or predictions.

As a baseline for measuring the performance of the RNN predictions, a naive prediction is made. The naive prediction is that the final data point in the input data is repeated for the future values. Hence, the naive estimate assumes that the SOH at any point in the future ($t > 0$) is the same value as the SOH at time zero ($t = 0$).

Looking at the predictions in Fig. 11, it seems that the RNN is capable of capturing the overall trends as well as the magnitude of the series. For stacks 13, 24, 70, 79, 98, 106, and 112 the predictions fit very well. Whereas for stacks 84 and 96, the RNN struggles a little to make an accurate prediction.

To compare the performance of the RNN to the naive predictions, the absolute errors of both to the true values are computed. The means of the stack prediction errors of the two methods are depicted in Fig. 12 with the 95% confidence interval indicated by the error bars. For the first prediction, the mean error for the naive and RNN predictions are similar at 0.035 and 0.017, respectively. For the following predictions, however, the RNN performs significantly better at around 30%-40% the mean

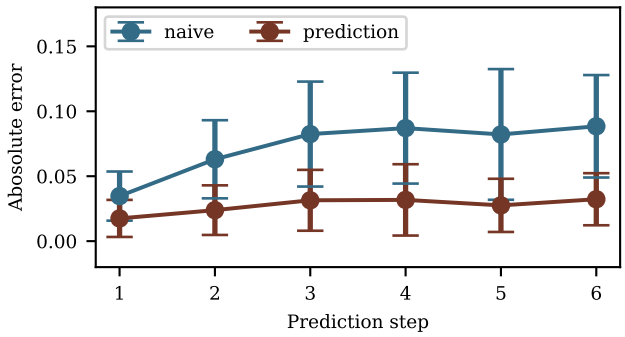


Fig. 12. Prediction error on the validation data of the naive baseline prediction and the recurrent neural network prediction. The error bars show the 95% confidence interval.

error of the naive predictions. At the six month prediction, the RNN prediction error is 0.032 - 64% lower than the 0.088 error of the naive prediction. It should be noted, that there is some overlap of the confidence intervals.

## VII. Long-Term State of Health Prediction

In this section, the trained RNN model will be used to make long-term predictions on the testing data. The RNN model is constructed to make

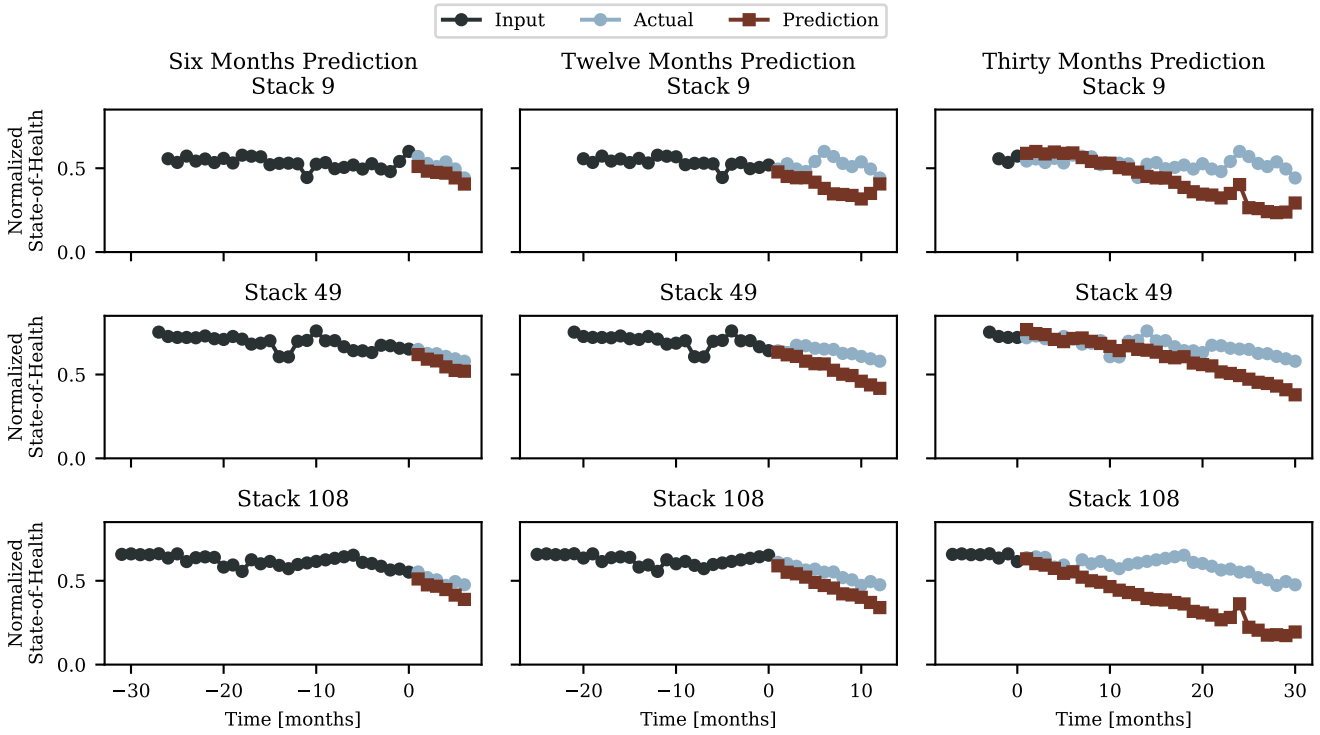


Fig. 13. Prediction results for the test stacks with a six, twelve, and thirty month prediction horizon, from left to right, respectively.

monthly predictions six months into the future. To achieve longer prediction horizons, the six months predictions are appended to the original input and used as a new input to the RNN which again predicts six months into the future, making a total of twelve predictions from the starting point. This process can be continued indefinitely. Predictions are made using this approach for each of the test stacks at different prediction horizons.

The number of samples of the test stacks are 33, 34, and 38, respectively. Therefore, the maximum prediction horizon, which can be verified with the sample data, is 30, i.e. five predictions of six months each.

Firstly, a prediction horizon of six months is calculated using the first 27, 28, and 32 samples for the input, leaving six samples for verification for each of the stacks. The results are shown in the left column of Fig. 13.

For prediction the SOH twelve months into the future, the first 21, 22, and 26 samples, respectively are used for the initial input to the RNN, leaving twelve samples for verification. The RNN predicts the SOH six months into the future. These six predictions are then appended to the initial input data, so that the new input to the RNN is the initial input samples as well as the first six predictions. The RNN then computes six new predictions, so that twelve predictions are made in total. The result is shown in the center column of Fig. 13.

The same approach is used for prediction horizons of 18, 24, and 30 months, i.e. three, four, and five iterations of RNN predictions, respectively. The result of the 30 month prediction horizon is shown in the right column of Fig. 13.

The absolute error at the end of each prediction horizon for each stack is calculated. For stack 9, the six month and twelve month prediction errors are very low, at around 0.069 and 0.068, respectively. The eighteen, twenty-four, and thirty months error all lie steadily between 0.225 and 0.384. The prediction error in stack 49 starts at 0.113 for the six month horizon. For the longer horizons, the error lies between 0.307 and 0.393. The prediction error of stack 108 has a bit more variation than the other two stacks. At six months the error is 0.166 and at twelve months it rises to 0.258 before falling again at eighteen months to 0.113. It increases again for the twenty-four and thirty month horizons.

The prediction errors of each stack for each prediction horizon are summarized in Table III along with the mean prediction error for each of the

TABLE III
PREDICTION ERRORS FOR TEST STACKS FOR PREDICTION HORIZONS FROM SIX TO THIRTY MONTHS.

| Prediction horizon | Stack 9 | Stack 49 | Stack 108 | Mean |
|---|---|---|---|---|
| 6 | 0.069 | 0.113 | 0.166 | 0.116 |
| 12 | 0.068 | 0.307 | 0.258 | 0.211 |
| 18 | 0.225 | 0.393 | 0.113 | 0.243 |
| 24 | 0.384 | 0.382 | 0.420 | 0.395 |
| 30 | 0.284 | 0.381 | 0.535 | 0.400 |
| Mean | 0.206 | 0.315 | 0.298 | 0.273 |

prediction horizons.

## VIII. CONCLUSION

Periodic self-test data has been used to estimate state of health (SOH) values for fuel cell stacks. The data contained voltage, current, and temperature measurements from fuel cell stacks operating in field conditions.

A long short term memory (LSTM) recurrent neural network (RNN) was trained on the SOH data to predict values six months into the future. The RNN was trained for 200 epochs where the mean absolute error loss function was reduced to approximately 0.02 on validation data, with a mean squared error of approximately 0.0005.

The prediction accuracy on the validation data was compared to a naive prediction, where the last SOH value was simply forecast in a repetitive manner. The RNN method showed significant improvement compared to the naive method, with some uncertainty.

Test data was used to test the RNN predictions at longer predictions horizons, by iteratively using RNN predictions in the input of the same RNN. The predictions were made from six to thirty months, in intervals of six months. The error remained relatively low for the longer prediction horizons, although the results vary from stack to stack.


[2] M. Bressel, M. Hilairet, D. Hissel, and B. Ould Bouamama, "Remaining Useful Life Prediction and Uncertainty Quantification of Proton Exchange Membrane Fuel Cell Under Variable Load," *IEEE Transactions on Industrial Electronics*, vol. 63, DOI 10.1109/TIE.2016.2519328, no. 4, pp. 1–1, 2016. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7386653


## REFERENCES


[1] M. J. Vasallo, J. M. Andújar, C. García, and J. J. Brey, "A Methodology for Sizing Backup Fuel-Cell / Battery Hybrid Power Systems," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 6, pp. 1964–1975, 2010.

[3] R. Ma, E. Breaz, C. Liu, H. Bai, P. Briois, and F. Gao, "Data-driven Prognostics for PEM Fuel Cell Degradation by Long Short-term Memory Network," in *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*, DOI 10.1109/ITEC.2018.8449962, pp. 102–107. IEEE, Jun. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8449962/

[4] T. Kim, H. Oh, H. Kim, and B. D. Youn, "An Online-Applicable Model for Predicting Health Degradation of PEM Fuel Cells with Root Cause Analysis," *IEEE Transactions on Industrial Electronics*, vol. 63, DOI 10.1109/TIE.2016.2586022, no. 11, pp. 7094–7103, 2016.

[5] C. Jeppesen, S. S. Araya, S. L. Sahlin, S. Thomas, S. J. Andreasen, and S. K. Kær, "Fault detection and isolation of high temperature proton exchange membrane fuel cell stack under the influence of degradation," *Journal of Power Sources*, vol. 359, DOI 10.1016/j.jpowsour.2017.05.021, pp. 37–47, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378775317306547

[6] N. Fouquet, C. Doulet, C. Nouillant, G. Dauphin-Tanguy, and B. Ould-Bouamama, "Model based PEM fuel cell state-of-health monitoring via ac impedance measurements," *Journal of Power Sources*, vol. 159, DOI 10.1016/j.jpowsour.2005.11.035, no. 2, pp. 905–913, 2006.

[7] T. Sutharssan, D. Montalvao, Y. K. Chen, W. C. Wang, C. Pisac, and H. Elemara, "A review on prognostics and health monitoring of proton exchange membrane fuel cell," *Renewable and Sustainable Energy Reviews*, vol. 75, DOI 10.1016/j.rser.2016.11.009, no. November 2015, pp. 440–450, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.rser.2016.11.009

[8] B. Dolenc, P. Boškoski, M. Stepančič, A. Pohjoranta, and . Juričić, "State of health estimation and remaining useful life prediction of solid oxide fuel cell stack," *Energy Conversion and Management*, vol. 148, DOI https://doi.org/10.1016/j.enconman.2017.06.041, pp. 993–1002, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196890417305861

[9] S. D. Sonderskov, M. J. Swierczynski, and S. Munk-Nielsen, "Lifetime prognostics of hybrid backup power system: State-of-the-art," in *2017 IEEE International Telecommunications Energy Conference (INTELEC)*, vol. 2017-Octob, DOI 10.1109/INTLEC.2017.8214199, pp. 574–581. IEEE, Oct. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8214199/

[10] S. D. Sønderskov, "Monitoring Performance Indicators of PEM Fuel Cell Backup Power Systems," in *2017 IEEE International Telecommunications Energy Conference (INTELEC)*. Torino, Italy: IEEE, 2018.

[11] A. L. Dicks and D. A. J. Rand, *Fuel Cell Systems Explained*. Chichester, UK: John Wiley & Sons, Ltd, Apr. 2018. [Online]. Available: http://doi.wiley.com/10.1002/9781118706992

[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, DOI 10.1162/neco.1997.9.8.1735, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[13] F. Chollet and Others, "Keras," \url{https://keras.io}, 2015. [Online]. Available: https://keras.io

[14] 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 {P}ython," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] D. Kingma and J. Ba, "Adam: a method for stochastic optimization (2014)," *arXiv preprint arXiv:1412.6980*, vol. 15, 2015.