State-of-the-art in individual residential energy demand forecasting

Elias Baumann (587775) , Josef Lorenz Rumberger (587961)

September 2018

Abstract

With the rise of renewable energy and single households implementing their own energy sources, energy grids are becoming more and more complex. Therefore dynamic and intelligent management of energy demand in residential areas is required in order to efficiently use the available power for different households. In the past, focus of energy demand forecasting was mainly done on an aggregate level and usually only considering the next few steps. With the widespread implementation of smart grids and smart meters, more granular data became available. Since numerous energy providers depend on long term information when deciding when to start and stop power plants, long sequences need to be forecast accurately. In this work, we review the work of Marino et al. (2016) and replicate as well as evaluate their implementation of forecasting single residential household energy demand over long periods of hourly and minutely resolution.

1 Introduction

Forecasting energy demand accurately has been the electricity suppliers' main problem for many decades. Because storing the energy is rather expensive due to the unavoidable energy loss inherent in the available technologies. Additionally with the rise of inexpensive local renewable energy generators, local grids are fluctuating strongly and therefore forecasting the total demand is vital when making decisions about supplying additional power.Forecasts of such demand can be done for different time horizons: long, medium and short term. When looking at long- and medium-term forecasts, energy suppliers usually make decisions about required capacities in the future, whether or not new power plants or transmission systems are needed and about general adjustments considering the long term trend of demand. When day, hour or minute ahead (short-term) forecasts are considered, single power systems can be scheduled and adjusted (Lee et al., 1992). When looking at energy demand, the total demand can be split into sectors and be evaluated individually. Segmented data is often available for individual residential, industrial, commercial and transportation consumption units. The residential sector is accounting for a large part of the total consumption of the entire United States electricity demand with projections showing no reduction in the overall percentage (Hostick et al., 2012). Forecasting the residential sector accurately would therefore already very much benefit the overall energy efficiency by allowing for peak load shaving for that sector(Kong et al., 2017). To be able to forecast the residential sector more accurately, smart meters have been put into place over the last decade. Smart meters allow for automatic collection of individual household electricity demand with short measurement intervals (Depuru et al., 2011). In the US, almost 50% of households are expected to have a smart meter by the end of 2019 (Ghofrani et al., 2011). Making use of this individual household data, energy suppliers can create forecasts for specific buildings, small communities or towns as well as large scale forecasts. Individual forecasts become especially interesting as residential customers are starting to supply energy using renewable sources themselves which in turn increases the volatility of demand greatly (Aghaei and Alizadeh, 2013). These individual energy sources can also put more power into the grid than needed for the building and grids need to adapt to such effects. As a result, smart grids are proposed. Smart grids use individual level predictions to adapt to energy demand dynamically, both at micro and macro level. This smart demand response can make use of individual renewable energy sources, further increase efficiency benefiting both customers and producers and deal with the new effect of electric vehicles on the grid (Siano, 2014). Smart grids rely on multi-step ahead forecasts to be able to adjust to changes in demand quickly. However individual households have very high variance in day to day behavior resulting in complex patterns which need to be predicted. In the last two decades machine learning has gained strongly in popularity as it is able to learn patterns and predict future timesteps with many unkown parameters for horizons of different length (Mocanu et al., 2016). This work will look at past work on forecasting both aggregated and invididual residential energy demand, specifically focusing on machine learning approaches and will then try to replicate the work of Marino et al. (2016). The work is structured as follows: first, we will analyze current literature on demand forecasting and residential demand forecasting in chapter 2. Then we will explain the methodology necessary to understand the replicated work in chapter 3 and present our results in chapter 4. Finally we will discuss the implications and reasons for the acquired results and give an outlook on what else can be done to further improve residential forecasting in chapters 5 and 6.

2 Literature Review

2.1 Statistical methods for demand forecasting

In order to establish a baseline to compare machine learning approaches, we first analyze current papers on demand forecasting using traditional statistical methods.

Verdejo et al. (2017) make use of multiple state-of-the-art statistical methods for time series forecasting in order to predict aggregated demand for one feeder in Santiago de Chile, both 15 and 30 days ahead. They use a dataset of one residential electricity feeder in Santiago which reports aggregated energy demand on a daily level for multiple households. Data is available for 135 consecutive days starting August 1^{st}.The authors additionally consider the average daily temperature in Santiago as it influences both air conditioning and informs about lighting over the day. Furthermore they make use of the unit of account of chile as an economic indicator and holidays as well as saturdays and sundays as special days with different electricity demand behavior as exogenous variables. They validate the statistical significance of all variables for each model. Verdejo et al. (2017) use three different models: Multilinear regression, harmonic regression and an autoregressive model. Additional to that, they use harmonic regression to model the residuals and a stochastic differential equation (SDE) to model the remaining noise. To estimate residential demand using multilinear regression, all three exogenous variables are used together. Parameters are then estimated using ordinary least squares. In order to evaluate the model, R^2 is calculated to check for good fit, normality is tested via the Jarque Bera Test, auto-correlation is tested via the Durbin Watson Test and homoscedasticity is tested with the Breusch Pagan Test. In this instance, the model does not pass the auto-correlation test which Verdejo et al. (2017) relate to the chosen exogenous variables. They suggest taking more variables into account for better performance. By identifying key frequencies of the observed data, Verdejo et al. (2017) estimate a harmonic regression model taking only the daily temperature as an additional exogenous variable. Again they estimate the parameters using ordinary least squares and the same tests as for the multilinear regression model are applied. This model also fails the autp-correlation test and furthermore does not pass the Jarque Bera test for normality of the errors. However, they show that by adding the temperature variable, they can improve the model fit by 70%. Their third approach, a SARIMAX $(1, 1, 1) \times (0, 1, 1)$[1] model uses temperature and holidays as additional exogenous variables and is estimated using a maximum likelihood estimator. They show that this model passes tests for independence and normality and that the Akaike information criterion is particularly low, which hints at a good model specification. Their final model is the harmonic regression modeling the residuals and the model noise is described using the Ornstein-Uhlenbeck process SDE. The new model is estimated using quadratic variations and a uniform residual analysis is applied to determine goodness of fit and if the fitted observations originate from their proposed harmonic regression model. Using a visual analysis of a quantile to quantile plot they conclude that the observations stem from an Ornstein-Uhlenbeck process and therefore their model is valid (Verdejo et al., 2017). Their results show that all models can explain the observed behavior very well, however the models forecasting accuracy varies. They report that the multilinear regression cannot forecast long forecasting horizons accurately because of unknown behavior of explanatory variables like temperature. The same problem occurs when forecasting with the harmonic regression model and even adding the stochastic differential equation model does not improve forecasting performance significantly. In Verdejo et al. (2017)'s work only the SARIMAX model is consistently capable of forecasting with a low mean percentage error, however their below 1% mean percentage error results are questionable as not even their example plots reflect his kind of low error. This paper demonstrated that statistical models are able to accurately model observed data but lack in forecasting accuracy.

Ghofrani et al. (2011) take a regular statistical approach too, but also use kalman filters to do short term residential load predictions. They are assuming that the residential load is a sum of two components: Weather Load and Lifestyle Load, where weather load is a gaussian noise signal impacting the energy demand because of HVAC (Heating, Ventiliation, Air Conditioning). The lifestyle load is then based on individual behavior of households but is seen as a deterministic part which is then in turn modeled using a polynomial fit. By subtracting the polynomial from the observed load, a mean zero gaussian noise signal results. Ghofrani et al. (2011) tried to estimate this gaussian noise signal by determining its shaping filter using spectral analysis and finally the kalman filtering algorithm. The spectral analysis is done by fitting a gauss markov model to the random signal in order to make use of its spectral density function. Using the spectral density funtion, a state space model is derived which in turn is used as the baseline for their kalman filter algorithm. The Kalman filter algorithm basically tries to calculate an a-posteriori estimate to update an initial state variable used to calculate a prediction. Ghofrani et al. (2011) then use this model to predict the behavior of a single residential household one to four steps ahead on

[1](P=Autoregressive factor,d=order of differencing,q=Moving average factor) x (P=number of seasonal autoregressive (SAR) terms, D=number of seasonal differences, Q=number of seasonal moving average (SMA) terms)

15-minute resolution data. Their one step ahead forecast with high sampling rate achieves decent mean absolute percentage error results for one to four steps ahead forecasting. However it is in the interest of energy providers to estimate accurate forecasts longer than one hour ahead.

2.2 Machine Learning for demand forecasting

The papers using mathematical statistical methods to forecast demand seem to be somewhat limited when considering their forecast accuracies and the used data. Machine learning is often reported to be capable of learning complex patterns and being able to forecast accurately (Murphy, 2012). However this often happens at a loss of explainability (Boulesteix and Schmid, 2014). And of course according to the no free lunch theorem by Wolpert and Macready (1997), no technique will always perform better. Still we will now analyze papers using machine learning to do single household energy demand forecasts.

Bonetto and Rossi (2017) attempt to forecast energy consumption in residential households using lightweight machine learning approaches in order to support smart grids in decision-making. They compare different methods considering different dataset sizes and training time. They compare four different models: an ARMA model as baseline comparison, a support vector machine (SVM) based model, a non-linear autoregressive model (NAR) and finally a long short term memory neural network. The SVM based model consists of multiple SVMs trained in parallel each given data to predict one step into the future. In the next step all predictions are combined to create a multi-step ahead forecast. The non-linear autoregressive model is a recurrent neural network, meaning that it is a neural network using the output of the same network one time-step prior and the last n timesteps. The different layers of this neural network have non-linear activation functions and in this specific instance the network consists of one hidden layer with 40 neurons and is trained using the Levenberg-Marquardt optimization algorithm with bayesian weight regularization. Bonetto and Rossi (2017) then use another form of neural network, the long short term memory neural network which is an extension of the recurrent neural network in that it not only considers the last step output but retains a memory of multiple past network states outputs and forgets some parts of it. As this method will be the focus of this paper, a detailed explanation will follow in Chapter 3.3. Here Bonetto and Rossi (2017) use one hidden layer with 50 memory cells and an output layer with one neuron. The network is trained using the ADAGRAD algorithm for faster convergence. Both neural networks, similar to the SVM are trained h times to predict h timesteps into the future resulting in a weight and bias matrix. The dataset to compare their results is a single residential household dataset with minute resolution and attempt to forecast 120 minutes using a 30 minute sample. In their results, all machine learning approaches perform better than the baseline ARMA model. Both neural networks perform well in the first 40 minutes of the forecast period, but are then surpassed by the svm forecast. Their SVM performs significantly better over the remaining forecast leading Bonetto and Rossi (2017) to the conclusion that because the LSTM requires 100.000 samples compared to the 5000 samples of NAR and SVM, its lower error variance is ignored and a combination of NAR and SVM should be used. They propose to use the results of the NAR for the first 40 minutes and then switch to the SVM results.

Mocanu et al. (2016) again try to forecast the demand of a single residential household but focus on the use of conditional restricted Boltzmann machines (CRBM) and factored conditional restricted Boltzmann machines (FCRBM). Conditional restricted boltzmann machines consist of the elements of a restricted boltzmann machine (a bipartite graph of a visible layer and a hidden layer) with an additional conditional history layer. The weights of the history layer are applied unidirectionally from the history layer to the other two layers. Boltzmann machines, similar to neural networks, learn features by iteratively updating weights. However these layers learn probability distributions rather than values and the error is computed by using contrastive divergence in order to compare the actual probability distribution with the estimated one. The FCRBM adds two more layers: a style layer and a feature layer, however Mocanu et al. (2016) combine these layers and add only three-way interactions between layers. This combined layer is fed with different additional parameters useful for an accurate prediction. Mocanu et al. (2016) compare their boltzmann machines to an artificial neural network (ANN), a NAR, a SVM and a RNN echo-state network. They evaluate performance using root mean square error (RMSE) and predict multiple timesteps for 15-minute, one hour and one day forecasting horizons with one minute resolution data and extended their predictions to use 15 minute, one hour and one week to forecast even longer time horizons up to one year. They compute predictions for both aggregated active power as well as the sub-meter power levels available in the dataset. In general Mocanu et al. (2016) show that their boltzmann machine models are viable for multi-step ahead forecasting and in most cases they outperform the other models based on RMSE. Recurrent neural networks have similar performance when using minute resolution and SVMs seem to perform similarly when using lower, hourly resolution. Overall boltzmann machines are more robust for longer prediction horizons. They report that all their models could deliver predictions within low hundreds of milliseconds making all of them viable for short term load forecasting in residential homes. It should be noted that this paper aims to demonstrate the power of boltzmann machines, yet the neural network based models are very simple compared to the boltzmann models and could be constructed deeper and much more complex. It would therefore be necessary to compare their work against a paper trying to prove that neural networks are the best forecasting method.

2.3 LSTM based neural networks for demand forecasting

For the specific task of dealing with time-series and lagged data, long short term memory neural networks were developed (Hochreiter and Schmidhuber, 1997). These neural networks have been used as comparison in already discussed papers, but have not been the focus.

Kong et al. (2017) specifically focus on using LSTM neural networks to forecast both individual household as well as system level load. They also show inconsistencies among individual households using density based clustering to further illustrate the problem of individual household forecasting. In the first part of their work, Kong et al. (2017) explore their dataset which includes 10000 electricity customers over 4 years. Individual and system level daily load profiles are then clustered using density based spatial clustering of application with noise (DBSCAN) resulting in one cluster for the daily system level load over 90 days. The individual households however show multiple clusters and more than 20 and up to 80 outliers over the 90 day period. They therefore conclude that an abstraction of past behavior has to be learned and they propose to use LSTM neural networks to solve this task. Kong et al. (2017) build a LSTM using energy consumption sequences as input and additionally time day and day of week indices as well as holiday marks all scaled to be between zero and one. They use two hidden layers with 20 nodes each and train the network for 150 epochs without tuning parameters and using the Adam optimizer. Forecasting results are then compared to several approaches. Empirical mean and an approach based on the statistical distribution of energy consumption and the corresponding probability mass function to minimize the mean absolute percentage error (MAPE) are used as empirical approaches. Furthermore, machine learning models such as two different artificial neural networks with two hidden layers each, k-nearest neighbor regression with 20 neighbors and an extreme learning machine, which is a feed forward network with a single layer, are used. Lastly they compare their LSTM to an input selection scheme with hybrid forecasting (IS-HF). All models use data of 92 days with 30 minute measurement intervals and get two, six and twelve time step sequences as input (Kong et al., 2017). Their results show the the LSTM proved to be the best model for all tasks, with MAPE minimization being a viable alternative for household forecasts and ANNs being able to capture the latent structure of the dataset as well. Additionally they show that, by aggregating forecasts for all households, they can outperform the simple aggregated load forecast. Finally Kong et al. (2017) evaluate the internal state of their LSTM and show that it successfully captures different behavior of different weekdays, however it still requires energy consumption patterns to be somewhat consistent. Parameter tuning and different layer counts could further improve the LSTM performance, nevertheless they do show that with sufficient data, the LSTM can forecast energy demand accurately over multiple days.

2.4 Empirical Review

Using the information gained from the literature review, we focus on a specific LSTM implementation used in **Marino et al. (2016)**. Similar to Kong et al. (2017), they try to forecast individual household demand using LSTM neural networks. Specifically two architectures are proposed: A basic LSTM which predicts a single step ahead using a sequence of previous timesteps as input. Multi-step ahead prediction can then only be done by using the output of the previous timestep as new input for the next step. To improve upon this model, they add lag to their data such that the model does not just learn a naive mapping from input to output. The second architecture uses a sequence to sequence encoder and predicts an entire sequence at once. The model additionally uses the time date variables as input for the prediction sequence. Marino et al. (2016) forecast individual residential household demand on minutely and hourly level. When forecasting multiple days on hourly resolution or multiple hours on minute resolution they show that their complex sequence to sequence encoder LSTM performs best. In order to validate their findings, this work will re-implement their work and evaluate it as well as compare it to some standard methods to forecast multivariate timeseries. To explain the architecture and models used, we will now introduce important theoretical concepts.

3 Methodology

In this paper sequential input data is used. A subset X_s of the complete sequence $X = (x_1, ..., x_t)$ is defined to make a point prediction for the electricity demand at time $t + h$, where h is the forecasting horizon. Forecasts are conducted for horizons e.g. from 1 up to 120 minutes. When forecasting, essentially a function $f(X_s, W)$ is searched that minimizes the cost function $J(X, \hat{X})$:

$$\begin{aligned} &\min J(X,\hat{X}) = \sum_{t=1+h}^{T-h} (x_{t+h} - \hat{x}_{t+h})^2 \\ &\text{s.t } f(X_s, W) = \hat{x}_{t+h} \end{aligned} \tag{1}$$

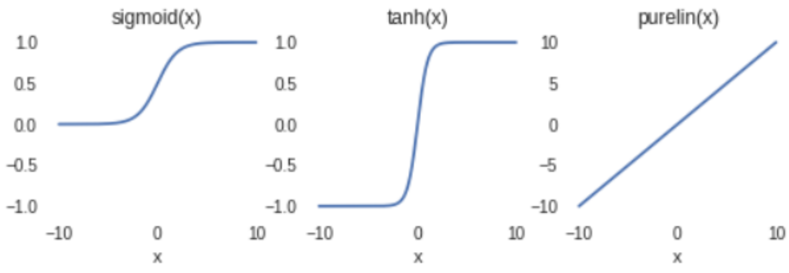
The cost function is defined as the sum of squared differences between the predicted value $\hat{x}_{t+h}$ and the realization x_{t+h}. .

3.1 Artificial Neural Networks (ANNs)

Just like their biological counterparts, ANNs consist of a number of layers L, each layer l contains a number of neurons $M^{(l)}$. An artificial neuron is a simplified mathematical model of a neuron cell from an biological entity. Some input vector $X = (x_1, ..., x_t)$ is represented to the neuron, in the first step the input gets weighted with $W = (w_1, ..., w_n)$ and bias b is added:

$$\begin{aligned} z_m^{(l)} &= \sum_{i=1}^{t} w_{m,i}^{(l-1)} \cdot x_i + b_m^{(l-1)} = W_m^{(l-1)T} \cdot X + b_m^{(l-1)} \\ a^{(l)} &= F(z^{(l-1)}) \end{aligned} \tag{2}$$

then an activation function $F(.)$ is applied on the weighted input $z^{(l)}$. Activation functions typically do a non-linear transformation of the input values (except purelin, which is only used in the last layer). If a vector is the input, the activation function will transform the vector elemen-twise. The ANNs applied in this paper use the sigmoid (σ), hyperbolic tangent (tanh or tn) and linear (purelin) activation functions:



$$\begin{aligned} \sigma(z^{(l)}) &= \frac{1}{1+e^{-z^{(l)}}} \\ tn(z^{(l)}) &= \frac{1}{1+e^{-2 \cdot z^{(l)}}} \\ purelin(z^{(l)}) &= z^{(l)} \end{aligned} \tag{3}$$

With the non-linearity introduced by these functions, a set of neurons can approximate non-linear functions by adjusting parameters $W_m^{(l)}$ and b_m. Typically more than a single neuron is present in each layer of an ANN. The first equation of (2) can be generalized for all neurons $M^{(l)}$ in a layer by concatenating the weights $W^{(l)} = (W_1^{(l)}, ..., W_m^{(l)})$ and biases $B^{(l)} = (b_1^{(l)}, ..., b_m^{(l)})$. So the activations for all neurons in a layer $A^{(l)} = (a_1^{(l)}, ..., a_m^{(l)})$ can be computed via equations (4).

$$\begin{aligned} Z^{(l)} &= W^{(l-1)T} \cdot X + B^{(l-1)} \\ A^{(l)} &= F(Z^{(l)}) \end{aligned} \tag{4}$$

The output of the first layer is fed as the input into the next layer in a feed-forward ANN, which is the simplest case. So instead of vector X later layers get $A^{(l-1)}$ as the input for equation (4), so the equation becomes

$$\begin{aligned} Z^{(l)} &= W^{(l-1)T} \cdot A^{(l-1)} + B^{(l-1)}, \text{for } l \neq 1 \\ A^{(l)} &= F(Z^{(l)}) \end{aligned} \tag{5}$$

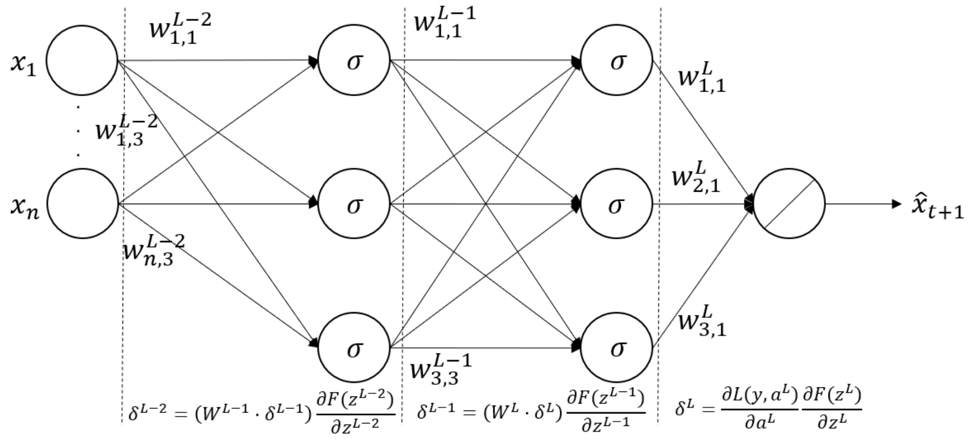
The limitations for ANNs for time series predictions arise when using longer input subsets X_s. For every additional element of the input vector, another M^1 weights (i.e. connections) are introduced for the first layer neurons. That is why long input data sequences become prohibitively costly from a computational view point (Goodfellow et al., 2016). Furthermore ANNs just allow for fixed size input sequence length, whereas variable length might be desirable for certain applications. These shortcomings led to the development of recurrent neural networks.

3.1.1 The Back-Propagation of Errors Algorithm applied on ANNs

A neural network learns through weight optimization via the gradient descent algorithm. The algorithm steps down the gradient, in order to find a local minimum (or a global minimum for convex optimization problems). A loss function $L(x_{t+h}, \hat{x}_{t+h} = a^L)$ that is object to optimization is defined ,then the partial derivative of that function with respect to each of the parameters in the model is computed, for an ANN it is:

$$\frac{\partial L(x_{t+h}, a^L)}{\partial W^l} = \frac{\partial L(x_{t+h}, a^L)}{\partial a^l} \frac{\partial a^l}{\partial z^l} \frac{\partial z^l}{\partial W^l} = \delta^l \frac{\partial z^l}{\partial W^l} \tag{6}$$

Before one can do this for all weights in all layers, the partial errors δ^l for every layer l of the network must be computed. This is done by the back-propagation of errors algorithm proposed by Rumelhart et al. (1986). The partial errors for the l-th layer of a feed forward network are denoted δ^l and computed as shown in figure 1



x_1
x_n
$w_{1,1}^{L-2}$
$w_{1,3}^{L-2}$
$w_{n,3}^{L-2}$
$w_{1,1}^{L-1}$
$w_{3,3}^{L-1}$
$w_{1,1}^{L}$
$w_{2,1}^{L}$
$w_{3,1}^{L}$
σ
$\hat{x}_{t+1}$
$\delta^{L-2} = (W^{L-1} \cdot \delta^{L-1}) \frac{\partial F(z^{L-2})}{\partial z^{L-2}}$
$\delta^{L-1} = (W^{L} \cdot \delta^{L}) \frac{\partial F(z^{L-1})}{\partial z^{L-1}}$
$\delta^{L} = \frac{\partial L(y, a^L)}{\partial a^L} \frac{\partial F(z^L)}{\partial z^L}$

Figure 1: Delta Rule applied on an ANN

Hochreiter (1998) found that the error gradient becomes smaller as it flows backwards through the network, resulting in faster learning for layers near the output and extremely slow learning or stagnation for layers near the input. The gradients become smaller as they flow backwards, because they are multiplied with parameters smaller than one. Since the weights are often initialized by a normal distribution with zero mean and unit variance, the weights are usually smaller than one. Furthermore the derivative of the sigmoid activation function is bounded between $[0, 0.25]$ and therefore the term $w^l \cdot \frac{\partial \sigma(z^l)}{\partial z^l}$ is often absolute smaller than 0.25. To visualize this further, this is how to calculate each δ^l for a 4 layer neural network:

- $\delta^L = (a^L - y) \cdot \frac{\partial f(z^L)}{\partial z^L}$
- $\delta^{L-1} = w^L \cdot \frac{\partial f(z^{L-1})}{\partial z^{L-1}} \cdot \left[(a^L - y) \cdot \frac{\partial f(z^L)}{\partial z^L}\right]$
- $\delta^{L-2} = w^{L-1} \cdot \frac{\partial f(z^{L-2})}{\partial z^{L-2}} \cdot \left[w^L \cdot \frac{\partial f(z^{L-1})}{\partial z^{L-1}} \cdot \left[(a^L - y) \cdot \frac{\partial f(z^L)}{\partial z^L}\right]\right]$
- $\delta^{L-3} = w^{L-2} \cdot \frac{\partial \sigma(z^{L-3})}{\partial z^{L-3}} \cdot \left[w^{L-1} \cdot \frac{\partial \sigma(z^{L-2})}{\partial z^{L-2}} \cdot \left[w^L \cdot \frac{\partial \sigma(z^{L-1})}{\partial z^{L-1}} \cdot \left[(a^L - y) \cdot \frac{\partial \sigma(z^L)}{\partial z^L}\right]\right]\right]$

The deltas computed as shown above are then used to carry out the parameter updates:

$$\begin{aligned} W^1_{new} &= W^1_{old} - \eta\delta^l X \\ W^l_{new} &= W^l_{old} - \eta\delta^l a^{l-1}\ , l \neq 1 \\ b^l_{new} &= b^l_{old} - \eta\delta^l \end{aligned} \tag{7}$$

With a learning rate η, that is applied to take smaller steps down the gradient. Typically η is in the range of $(0.1, 0.0001)$. The first line shows the general update rule, while the second and third show the rule applied on the first layer and on any other layer. The weights of the first layer get updated by subtracting the product of delta with the input X and the learning rate η. The weights of the consecutive layers are updated by subtracting delta multiplied with η and its former input a^{l-1}. The bias is updated according to the rule stated in the last line of equation 7.

3.2 Recurrent Neural Networks (RNNs)

RNNs incorporate a series of former predicted values $\hat{x}$ or a cell state into equation 1, so the function whose parameters W are estimated becomes:

$$\hat{x}_{t+h} = f(X_s, \hat{X}_t, W) \tag{8}$$

where $\hat{X}_t = \{\hat{x}_t, ..., \hat{x}_{t-n}\}$ contains the last predictions. This allows for input sequences of variable length and furthermore the network can process longer sequences than an ANN at the same computational costs. The reason for this is parameter sharing. Unlike an ANN where one would have separate parameters for each value of the input sequence, the RNN takes its predecessing predictions as an input and learns a parameter for it. The last prediction $\hat{x}_{t+h}$ is used by the network like a lossy summary of former task relevant information of the input sequence up to time t. It is obviously lossy, because the information contained in the input sequence up to t is encoded in the output, which could be a scalar $\hat{x}_t$ or a sequence $\hat{X}_{t,...,t-n}$. This makes RNNs invariant against perturbations inside the input sequence between parts that contain encoded information and parts that do not. ANNs would need to learn the same rule how to extract the relevant information for every input node if a certain type of relevant information can occur at any part of the sequence. RNNs on the other hand would just learn the rule once and then apply it whenever that type of information occurs. This lets RNNs learn reoccurring complex patterns faster and with better generalization than ANNs (Goodfellow et al. (2016)). In fact RNNs are turing complete, so every possible program can be modeled with it (Siegelmann (1995)). An RNN constructed from the above explained ANN is shown in figure 2. The regular inputs $x_1, \ldots, x_t$ change, because a sliding window of values is fed into the network and the prediction is always a single step prediction. So based on $x_1, \ldots, x_t$ and $\hat{x}_t$ the next element of the sequence x_{t+1} is predicted and denoted as $\hat{x}_{t+1}$. The last prediction of the network is always fed into the network for the next prediction. Therefore figure 2 shows the same network at three different points in time, which is called the unrolled representation of the network. There exist way more advanced forms of RNNs, some have recurrent connections on every node, so every node gets information about its past activations or the recurrent connections sum up cell states of all former activations of a specific node. ANNs and the type of RNNs that is explained so far both suffer from slower optimization of weights for layers deep in the network (near the input nodes). The more layers you add, the smaller is the gradient that arrives at the first hidden layer

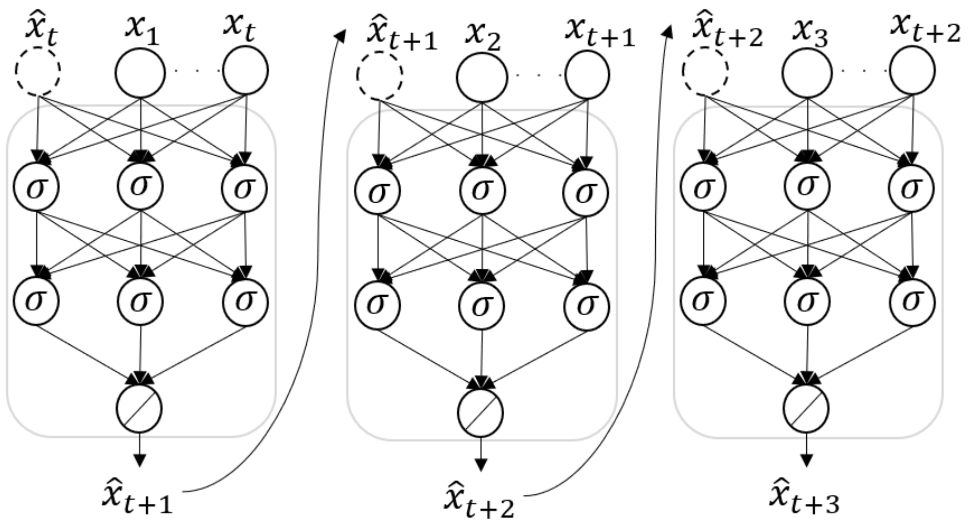


Figure 2: A simple RNN unrolled at different points in time

and it eventually comes close to zero. This is called the vanishing gradient problem. Fortunately Long Short Term Memory (LSTM) Networks, a modification of RNNs, can overcome these limitations (Hochreiter (1998)).

3.2.1 The Back-Propagation of Errors Algorithm applied on RNNs

The fundamental idea of weight optimization via gradient descent as outlined by equation 6 applies here too. But the application of the back-propagation algorithm and therefore the computation of the deltas differ. Unlike an feedforward ANN, every prediction made by an RNN is based on the according input data and the state of the network at that time, approximated by the last prediction $\hat{x}_{t-1}$. Therefore the error must be back-propagated through multiple states of the network, thus the algorithm is called back-propagation through time. Figure 3 shows the forward pass in black arrows and the back-propagation steps of the error computed for the last output $\hat{x}_{t+3}$ with light red arrows. The RNN cell can be composed of arbitrary network architectures. In figure 2 the cell is composed of a feedforward network with the same architecture as shown in figure 1. In this example the simplest case of a single neuron with activation function F is choosen. A loss function $L(x_t, \hat{x}_t)$ is defined and applied

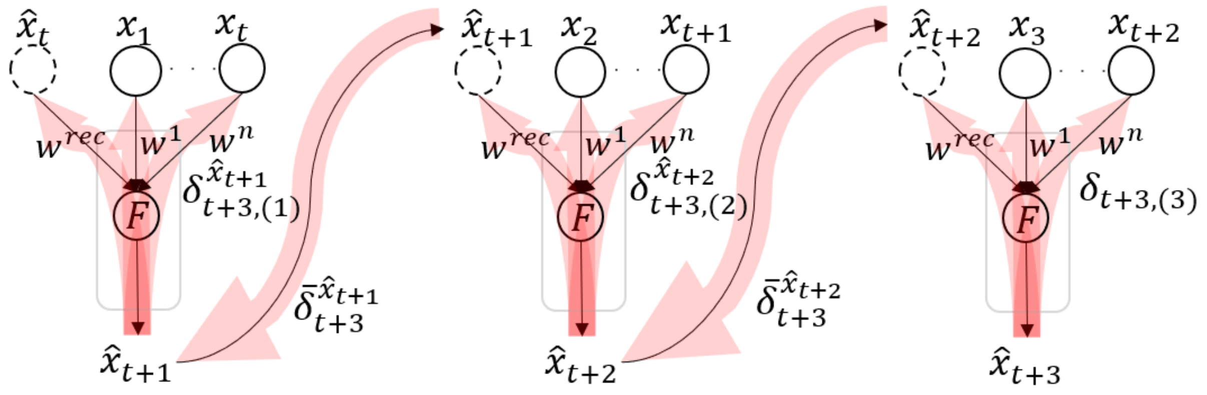

$\hat{x}_t$
x_1
x_t
w^{rec}
w^1
w^n
$\delta^{\hat{x}_{t+1}}_{t+3,(1)}$
F
$\bar{\delta}^{\hat{x}_{t+1}}_{t+3}$
$\hat{x}_{t+1}$
x_2
x_{t+1}
$\delta^{\hat{x}_{t+2}}_{t+3,(2)}$
$\bar{\delta}^{\hat{x}_{t+2}}_{t+3}$
$\hat{x}_{t+2}$
x_3
x_{t+2}
$\delta_{t+3,(3)}$
$\hat{x}_{t+3}$

Figure 3: Delta Rule for a single backward pass of the error of $\hat{x}_{t+3}$ in an RNN

on the last prediction made by the network. Then the derivative of the loss with respect to the weight vector

$W = (w^{rec}, w^1, \dots, w^n)$ is taken (remember: $a = F(z)$):

$$\frac{\partial L(x_{t+3}, \hat{x}_{t+3})}{\partial W} = \frac{\partial L(x_{t+3}, \hat{x}_{t+3})}{\partial a_{t+3}} \frac{\partial a_{t+3}}{\partial z_{t+3}} \frac{\partial z_{t+3}}{\partial W} = \delta_{t+3} \frac{\partial z_{t+3}}{\partial W} \tag{9}$$

with delta as a vector $\delta_{t+3,(3)} = (\delta^{\hat{x}_{t+2}}_{t+3,(3)}, \delta^{x_3}_{t+3,(3)}, \dots, \delta^{x_{t+2}}_{t+3,(3)})$. The next step is to propagate the partial error based on the value of $\hat{x}_{t+2}$ back to the very same network at the timestep before. This is computed as $\bar{\delta}^{\hat{x}_{t+2}}_{t+3} = w^{rec}\delta^{\hat{x}_{t+2}}_{t+3,(3)}$. This $\bar{\delta}^{\hat{x}_{t+2}}_{t+3}$ is handled as a partial loss, so the regular barless delta used to adjust the weights is calculated as $\delta^{\hat{x}_{t+2}}_{t+3,(2)} = \bar{\delta}^{\hat{x}_{t+2}}_{t+3} \frac{\partial F(z_{t+2})}{\partial z_{t+2}}$. The same computations are then applied to get $\bar{\delta}^{\hat{x}_t+1}_{t+3}$ first and then compute $\delta^{\hat{x}_{t+1}}_{t+3,(1)}$. The resulting weight updates for all weights are in the end summed up and applied as shown in equation 7. The next step is to do the same steps for the losses generated by $\hat{x}_{t+2}$ and $\hat{x}_{t+1}$. The first cell state $\hat{x}_t$ has to be initialized somehow before one can compute the first forward pass. This initialization can be learned as a parameter of the model by calculating the partial error of the loss associated with $\hat{x}_t$ as $\bar{\delta}^{\hat{x}_{t+2}}_{t+3} = w^{rec}\delta^{\hat{x}_{t+1}}_{t+3}$ and then changing $\hat{x}_{t,new} = \hat{x}_{t,old} - \eta\bar{\delta}^{\hat{x}_{t+2}}_{t+3}$.

One can see that the vanishing gradient problem introduced in equation 3.1.1 applies here too, since the gradient coming from the loss produced by $\hat{x}_{t+3}$ runs through three derivatives of activation functions until it reaches the first state of the network. Therefore the network learns long range dependencies really slow and some dependencies may be to long to learn because the gradient would already be vanished.

3.3 Long Short Term Memory (LSTM) Neural Networks

Hochreiter and Schmidhuber (1997) introduced an advanced version of the recurrent neural network to solve the vanishing gradient problem. The basic principle of recurrency is kept, however each individual cell is more complex. Figure 4 shows the basic structure of an LSTM cell applied on the same example task that we used in figure 2 and 3. The activation function layers consist of neurons with activation functions as explained in equation 3.

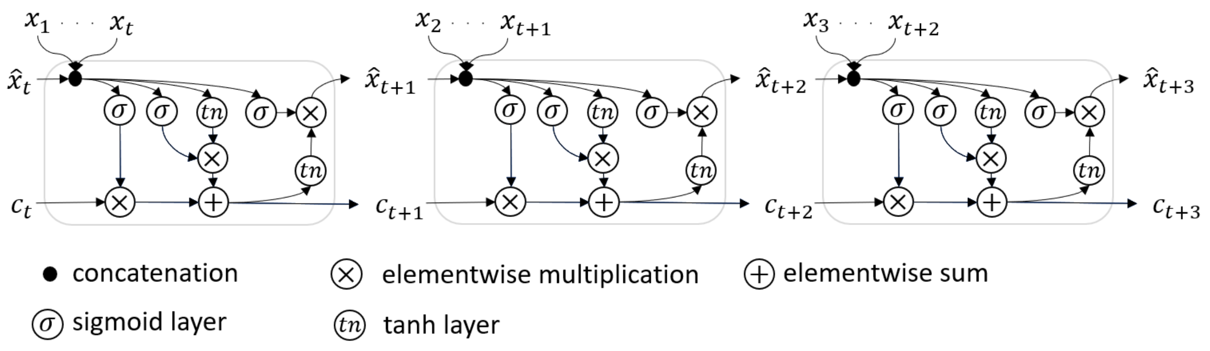


Figure 4: Long Short Term Memory Network

The network passes its last prediction $\hat{x}_t$ to its next state and uses it there as an input. Furthermore a cell state c_t is calculated and passed through the network too. The cell state gives the network the possibility to extract and learn information in sequences of arbitrary lengths, thus it is sometimes called the long term memory of the network. The upper image of figure 5 shows the operations inside the LSTM cell that alter the cell state and equations 10 show all equations governing the behavior of an LSTM cell. The inputs $x_1, \dots, x_t$ and the last prediction $\hat{x}_t$ are concatenated, forming the input stream. Then this stream is fed via the blue circuit to a layer of artificial neurons with sigmoid activation function (e.g. sigmoid layer). A sigmoid layer produces activations in the range $(0, 1)$.

By element-wise multiplying $(*)$ the activations with the cell state, part of the cell state is multiplied with values close to zero or zero. Thus the network can learn to forget parts of the cell state based on the concatenated inputs and the according weights in the sigmoid layer. That is why the multiplication operation is also called the 'forget gate'. The green circuits of the upper image show the part of the LSTM cell that writes information based on the inputs into the cell state. The sigmoid and tanh layers both get the concatenated inputs and produce activations. Remember the tanh activation function produces outputs in the range of $(-1, 1)$. The activations from both layers are multiplied, lowering and even canceling out some of the tanhs activations. Then the remaining activations are element-wise summed with the values of the cell state, which updates the cell state to c_{t+1}. The purple circuits in the lower image show the computations done to calculate the LSTMs output prediction $\hat{x}_{t+1}$. The updated cell state c_{t+1} is fed to a tanh layer, while the input stream is fed into a sigmoid layer. Thereafter both streams are element-wise multiplied to get the output prediction $\hat{x}_{t+1}$.

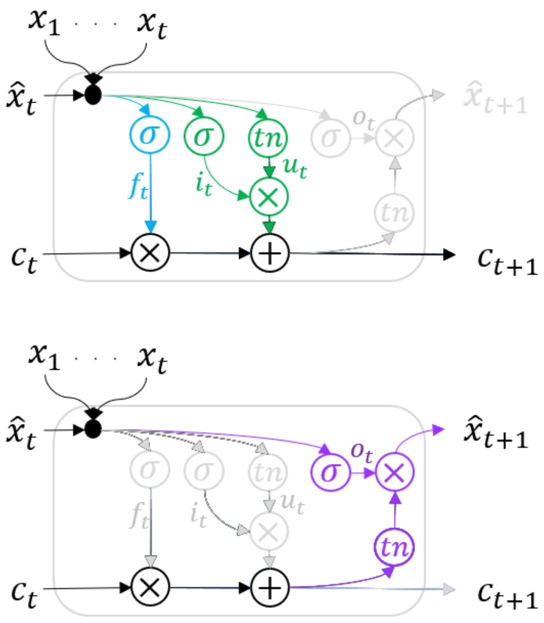


Figure 5: LSTM

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [\hat{x}_t, X_t] + b_f) \\
i_t &= \sigma(W_i \cdot [\hat{x}_t, X_t] + b_i) \\
u_t &= tanh(W_u \cdot [\hat{x}_t, X_t] + b_u) \\
o_t &= \sigma(W_o \cdot [\hat{x}_t, X_t] + b_o) \\
c_{t+1} &= f_t * c_t + i_t * u_t \\
x_{t+1} &= o_t * tanh(C_{t+1})
\end{aligned}
\tag{10}
$$

3.3.1 The Back-Propagation of Errors Algorithm applied on LSTM networks

As already mentioned, the architecture of the LSTM cell provides a depth independent gradient that is passed back through the former states of the network. The gradient of the current cell state with respect to the former cell state is calculated as follows:

$$
\begin{aligned}
\frac{\partial L(x_{t+1}, \hat{x}_{t+1})}{\partial c_{t+1}} &= \delta_{c_{t+1}} \\
\frac{L(x_{t+1}, \hat{x}_{t+1})}{\partial c_t} &= \frac{\partial L(x_{t+1}, \hat{x}_{t+1})}{\partial c_{t+1}} \frac{\partial c_{t+1}}{\partial c_t} = \bar{\delta}_t^{c_{t+1}} = \delta^{c_{t+1}} * f_t \\
\frac{\partial L(x_{t+n}, \hat{x}_{t+n})}{\partial c_t} &= \bar{\delta}_t^{c_{t+n}} = \delta_{c_{t+n}} \prod_{t=t}^{t=n} f_t
\end{aligned}
\tag{11}
$$

First the gradient of the loss with respect to the cell state at time $t + 1$ is computed, then this partial loss is used to calculate the partial loss of the cell state at time t. The third equation in 11 generalizes this behavior for all future cell states back to the first cell state at time t, Hochreiter and Schmidhuber (1997) call this the constant error carousel. The gradient flow is only hindered by the value of the forget gate f_t, which is near 1 if the cell state is propagated forwards and 0 if it is forgotten. So the gradient reaches as deep as it has to and allows learning of long term dependencies. The other gradients coming from $\hat{x}_t$ are calculated as in the RNN shown in figure 3 and the weight updates of the activation function layers inside the LSTM follow the same logic, they just have a higher gradient due to the constant nature of $\bar{\delta}_t^{c_{t+n}}$. Pascanu et al. (2013) give a more in depth explanation of the other gradients inside the LSTM cell for further studies.

3.3.2 Issues with a standard LSTM architecture

The standard LSTM still has a number of problems. For this example the biggest problem is the learning of a naive mapping. A usually very good prediction for the next time step is the exact value of the previous time step. The network learns this pass-through of the previous value and does not actually learn long term dependencies or patterns (Marino et al., 2016). Therefore, when introducing lag and forecasting multiple steps into the future, the basic LSTM network does not perform as well. To avoid this problem the historical data can be encoded as a vector such that the previous time step cannot be inferred as easily. On the other hand, when predicting multiple steps ahead, the target variable has to be lagged. Using this strategy however, dependencies over long periods are learned, but not over short periods. Both issues can be avoided when using sequence to sequence encoder LSTM networks, which will be introduced in the following chapter.

3.4 Sequence to sequence encoder LSTM NN

Marino et al. (2016) introduce an architecture for time series forecasting which uses an encoder-decoder architecture by combining two LSTM neural networks. To understand this architecture, the concept of encoders and decoders as well as auto-encoders should be understood.

3.4.1 Encoders, decoders, auto-encoders

Using encoders and decoders neural networks recently gained popularity because of its use in machine translation (Sutskever et al., 2014) but also found use in image segmentation (Badrinarayanan et al., 2015). Marino et al. (2016) apply the machine translation idea to their time-series problem.

Encoders try to create a hidden and smaller representation of the input data such that e.g. a picture is encoded in a different number of dimensions. This dimensionality reduced picture is then received by the **decoder** which tries to use the abstract representation to either predict some classification or new value or alternatively to reconstruct the original picture (Badrinarayanan et al., 2015). This encoding and decoding can be trained in an unsupervised manner if the goal is to simply reconstruct the original data as the network only compares its created reconstruction to the original input. The learned abstract representation is the important factor here, as this will prevent the network from learning any naive mappings or too simple patterns as the data has to be compressed to a smaller representation (Marino et al., 2016). Huang et al. (2007) refer to such encoder decoder network as feature extractors, having similar use to concepts such as principal component analysis. Figure 6 shows a very simple encoder decoder architecture with the goal of recreating the original input sequence $x_1, ..., x_n$ by encoding it in a lower number of neurons, e.g. h_1, h_2. This exact form of encoder decoder network is what is commonly referred to as an **autoencoder**. This basic encoder however has the limitation that input sequences always have to be the same length as output sequences. This can especially be an issue in fields such as machine translation as sentences do not necessarily have the same length in different languages (Sutskever et al., 2014). Therefore, they propose a new encoding decoding architecture called sequence to sequence encoding.

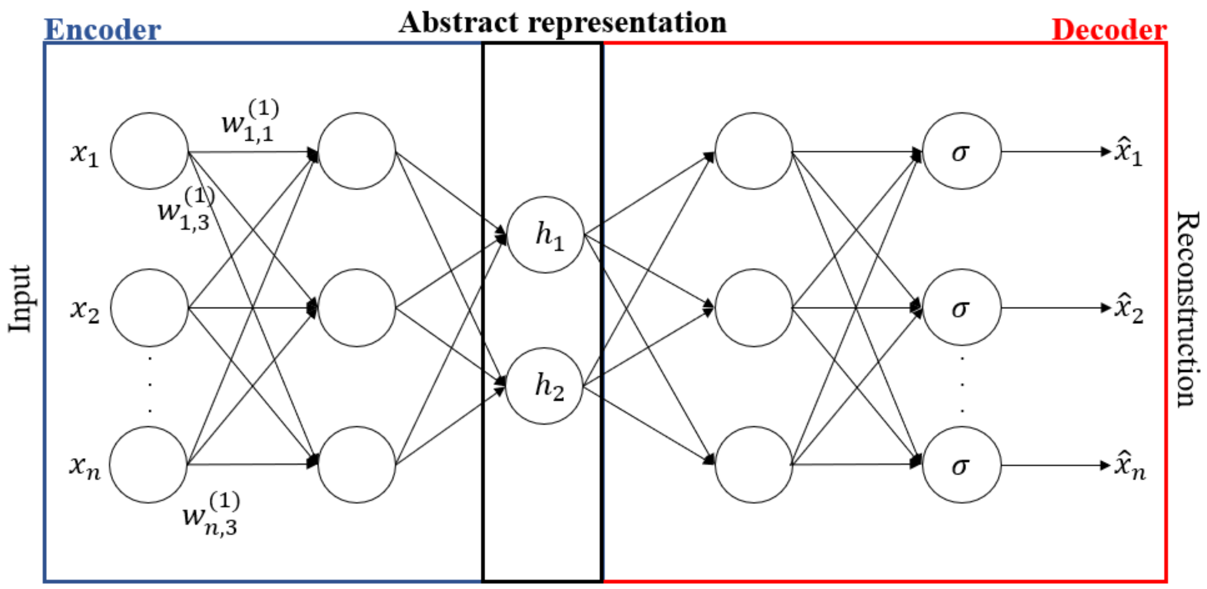


Figure 6: Simple encoder-decoder network to learn an abstract representation

3.4.2 Sequence to Sequence LSTMs

As LSTMs are capable of learning long term dependencies Goodfellow et al. (2016) , they can be used to create an abstract representation which include temporal dependencies. In Sutskever et al. (2014)'s architecture, this abstract representation v is a first LSTM networks last hidden state which encodes all information the network learned by autoencoding an input sequence of arbitrary length $x_1, ..., x_t$. A second decoder LSTM can then use the last hidden state v as initial state representation to predict another sequence. It can have a different output vector of arbitrary length $\hat{x}_1, ..., \hat{x}_t'$ as it only needs the hidden state as input (Sutskever et al., 2014). Using this architecture, Marino et al. (2016) try to forecast energy demand by using different length input and output sequences. Additionally they use temporal variables for both networks.

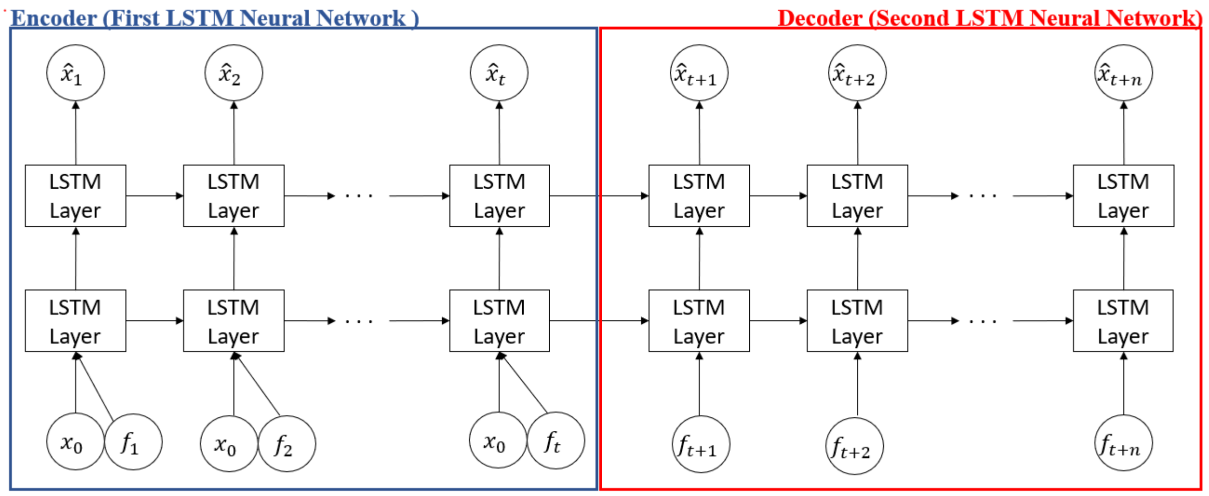


Figure 7: Sequence to Sequence architecture

Figure 7 shows their architecture which closely resembles the architecture of Sutskever et al. (2014). However the first difference is that the encoder tries to predict $\hat{x}_2, .., \hat{x}_{t+1}$ instead of just reproducing the input values $x_1, ..., x_t$. Furthermore, the decoder neither reproduces the original input but predicts the sequence $\hat{x}_{t+2}, ..., \hat{x}_{t+n}$ following the encoder prediction sequence. It is also important to note, that they use multiple LSTM layers in the encoder which then share their last hidden state with a corresponding LSTM layer in the decoder. $f_1, f_2, ..., f_t + n$ are additional inputs containing the timestamp or other additional data already available for the future. The original encoder decoder network can only be trained by training the entire network. However because the sequence to sequence architecture consists of two separate networks, Marino et al. (2016) first train the encoder separately and then with already trained weights, combine both networks to only fine tune the encoder weights and train the decoder weights. Even though the networks are separate, backpropagation of errors works rather similar as if it were one network. The shared last hidden state of the encoder network is adjusted by decoder and as both encoder and decoder access those values, the encoder receives these changes as well. There are two loss functions to minimize, the loss function of the encoder network and the loss function of the combined encoder decoder network. When training the networks together errors from both loss functions are used to adjust weights and states.

3.5 Data

The dataset contains 2075259 rows with values from December 2006 to November 2010 which corresponds to 47 Months. The data is taken from UCI Machine Learning repository and is called *Individual household electric power consumption Data Set* (Dheeru and Karra Taniskidou, 2017). The Dataset contains a number of variables, we will however only make use of *Global Active Power*, *Date* and *Time*. *Global Active Power* refers to the household global active power in kilowatt averaged over a minute. For different experiments the data was aggregated to different levels such as hourly or 15-minute interval data. Missing values are also present in the dataset. Consequently the values were interpolated linearly. These values do then not represent actual behavior but allow all models to work with the available data. The data was not normalized as the paper this work is based on does not normalize data either. Model performance was also not influenced significantly by adding this step and without normalization we can compare our results to the paper results.

4 Results

This work attempted to implement the proposed architecture by Marino et al. (2016) in order to forecast the energy demand of a single household. Data was split 75 / 25 training and testing data and LSTMs used another 20% of the training data for validation. In order to compare the performance of the sequence to sequence architecture, we compare the results to a basic LSTM, Support Vector Regression (SVR) and a simple linear regression model from *Scikit-Learn*. To actually compare the full performance of every model, it is necessary to train all models on all different possible lags within a forecast horizon. However as this process would take a very long time, we instead calculate the RMSE for models with specific interesting lags. The linear model was used with its base parameters feeding in the entire training data, except for the one minute resolution where we do not have the memory necessary when using the full set. Here we use 50%. The model uses an intercept as we do not center the data. The support vector regression model is fitted with base parameters as well. C, the error penality parameter, is set to 1.0 and epsilon, the distance in which no penality is applied, is set to $.1$. As training a SVR model takes a long time, we reduced the size of the original dataset by randomly sampling the full set. For the different time resolutions, $50\%, 12.5\%$ and 1% were used. The standard LSTM uses two LSTM layers, 50 neurons each, and a dense layer with a linear activation function. It is trained for 10 epochs with a batch size equal to the forecasting horizon to imitate the behavior of the sequence to sequence model. Finally the sequence to sequence model was also trained with two LSTMs with two layers having 50 neurons each. The encoder was trained for 5 epochs and the decoder for an additional 10 epochs. This results in overall longer training than the standard LSTM, however Marino et al. (2016) recommend to pretrain the encoder network. Using the residential household energy demand dataset we then forecast with every model.

4.1 Hourly Resolution

First, we consider hourly resolution where we attempt to forecast one week (168h). With this aggregation level, long sequences show more regular patterns and less noise making it easier to make good long term estimates. Table 1 presents the RMSE values for the different algorithms and models for specific time steps as well as the S2S Models RMSE values for that specific timestep in the sequence. For this hourly resolution data, all models

Model	1h	12h	24h	96h	168h
LM	0.56950	0.77835	0.69985	0.71964	0.70989
SVR	0.56030	0.72669	0.71833	0.72566	0.74034
LSTM	0.55106	0.70277	0.68966	0.71281	0.69862
Seq2Seq LSTM	0.53318	0.64362	0.65079	0.65750	0.67070

Table 1: Hourly Resolution RMSE for all Models for different Timestamps

are within similar error levels. All models perform very similar forecasting one step ahead (1h) with an RMSE of around 0.55. This could be an indicator that all models do a naive map from the input to the output value with the exception of the S2S Model which slightly improves over the other models with a 0.53 RMSE. As a naive mapping is no longer possible for the other timestamps, errors increase for all models. The difference between the 12h and 24h error are interesting as well, as for comparison models the 24h error is lower than the 12h error. This repetition does not however hold for 96h even though it would represent a daily repetition as well. This observation also does not hold for the S2S Model of which the error continually increases regardless. Figure 10 shows the error over the entire sequence.

Finally it seems that after a week both the LSTM and LM improve again which could be a reflection of a weekly pattern. Overall the S2S Model is a significant improvement over all other models with RMSE improvements of up to 17% over the linear model and 9% over the regular LSTM. To demonstrate the forecasting capabilities of the S2S model, Figure 8 shows the encoder and decoder sequence forecast. Analyzing this graph, we observe that the network has been able to capture the twin peak daily behavior of morning and evening power consumption as well as different behavior for different days. We also observe that the encoder does in fact not learn a naive map from input to output as the encoder forecast does not resemble a lagged original behavior. However the decoder LSTM seems unable to predict large spikes and other extreme values.

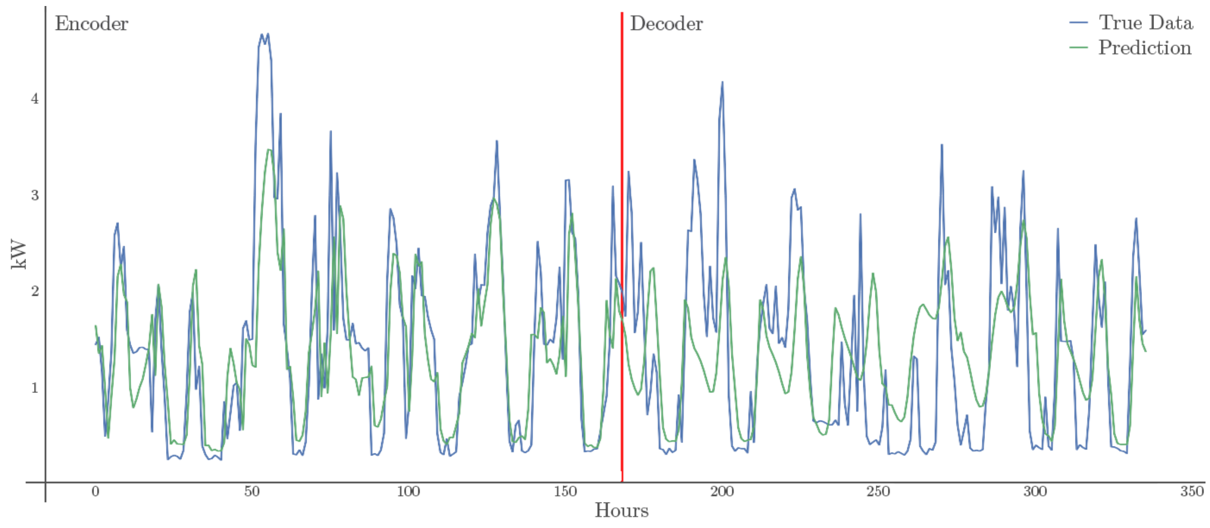


Figure 8: encoder and decoder sequence forecast sample

4.2 15 Minute Resolution

In this section, data with 15 minute resolution is analyzed. A sequence of 24h is considered which is the equivalent of 96 15-minute-intervals. The german electricity market uses these intervals as its shortest standardized time frame to trade electricity to balance out deviations. Energy from renewable sources has to be traded day-ahead according to german legislation. Since there are no perfect forecasts available for these energy producers, there the intraday market which these producers use to balance out the deviations of actual production from day-ahead forecasted production. The quarter hour intervals traded in this market can only be traded up to 30 min before they actually start. Therefore this resolution is of particular interest, especially the several hour ahead forecasts for the 15-minute-interval (Kiesel and Paraschiv, 2017). When inspecting the RMSE values for the different timestamps in Table 2, we see that for 15 Minute resolution, the overall error has increased except for the one step ahead forecast. All comparison models again have a comparatively low RMSE when considering the 24h forecast. Comparing the S2S Model only to the LSTM we see that the differences are smaller than for one hour and the regular LSTM even beats the S2S Model at forecasting one step ahead. For the remaining forecast period the errors are still significantly smaller for the S2S model making it again the best performer on 15 minute resolution. Considering again the RMSE curve of the 15 minute interval S2S Model in Figure 10 , the error seems to stay

Model	15min	1h	6h	12h	24h
LM	0.48854	0.74320	0.85919	0.86907	0.80262
SVR	0.52661	0.74023	0.83920	0.84034	0.80782
LSTM	0.48183	0.71723	0.78560	0.79398	0.78356
Seq2Seq LSTM	0.48904	0.69531	0.76865	0.77061	0.77232

Table 2: 15 Minute resolution RMSE for all Models for different Timestamps

almost constant after around 24 15 Minute intervals. Comparing this curve to Figure 10 the error seems to plateau earlier but stay at that level whereas the 1-hour error increases again after plateauing. Moreover the first incline is much steeper for the 1-hour curve than for the 15 minute curve.

4.3 One Minute Resolution

Finally, we consider the one minute resolution data in Table 3. Again the S2S model outperforms the comparison models, with the exception of forecasting one minute ahead, where it is beaten by both the regular LSTM and the linear model. In this high resolution data, the error is increasing for every model when the forecasted timestep is further in the future. The differences between the regular LSTM and the S2S LSTM are still significant particularly in the mid-section of the 120 minute sequence.

Model	1min	5min	30min	60min	120min
LM	0.22479	0.46382	0.74378	0.82921	0.89824
SVR	0.54243	0.63344	0.83040	0.85875	0.88222
LSTM	0.22788	0.46378	0.71224	0.78063	0.82223
Seq2Seq LSTM	0.24785	0.44524	0.68677	0.75572	0.81469

Table 3: 1 Minute resolution RMSE for all Models for different Timestamps

The error curve for one minute resolution in Figure 10 starts low and ends high but with low but continuous gradient. Interestingly this curve is the only strictly positive curve and it is likely that a longer forecast would lead to an even higher error. Finally, considering again the plot of a sequence forecast in Figure 9, we can see that the

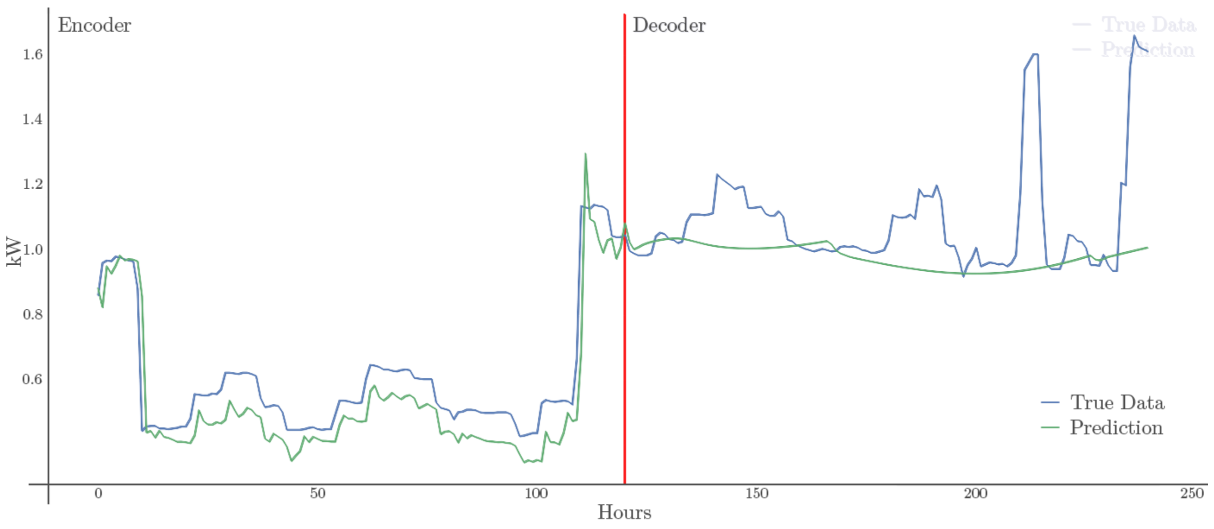


Figure 9: One minute resolution encoder and decoder sequence forecast sample

encoder forecast is somewhat naive as it resembles a one step lagged version of the original in some parts. We also see, that the decoder is not able to predict spikes and strong differences over time but instead approximates the general direction.

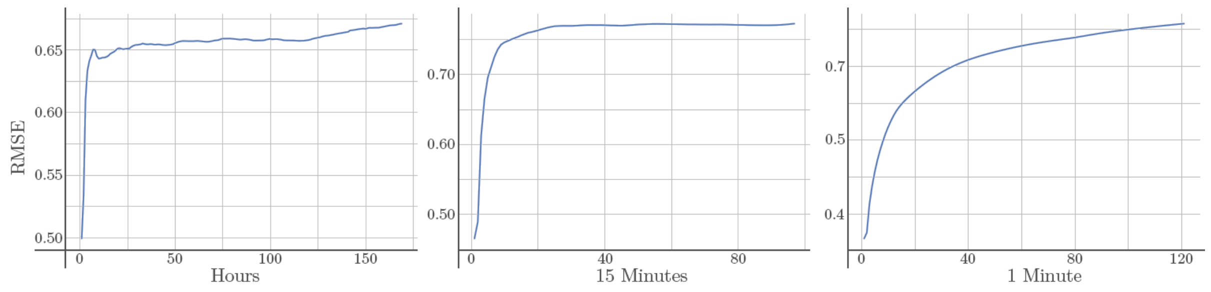


Figure 10: RMSE of S2S model over predicted sequence for different resolutions

5 Discussion

The proposed and evaluated sequence to sequence architecture works well for noise irregular single residential household data. The model shows impressive results both forecasting hourly and minutely resolution data even over long periods of time with significant error differences to the competing models. The model seems to capture both overarching behavior within the household as well as more nuanced effects such as the behavior on different weekdays or within certain hours. A reason for the higher error for a 1 minute ahead forecast of the S2S model could the error back-propagation from the decoder network. The regular LSTM stays closer to a direct input to output mapping, as does the linear model. Overall this S2S architecture should not be used if the only goal is to forecast only one step ahead. The S2S Model has an important advantage over models like the SVR and LSTM as to achieve any kind of comparable performance, for both models one would have to train one model for every step of the forecast sequence. While currently the training of a single minute resolution S2S Model takes multiple hours, training the other models e.g. 120 times for a 120 minute horizon, would take an order of magnitude longer. In a streaming scenario where the model continually forecasts the next sequence, the forecasts would also need to be computed 120 times. A intuitive behavior can be observed when looking at specific intervals like 24h and one week, as for both cases, the comparison models have better accuracy here than for an irregular point in time such as 12h or 96h. This hints at typical household energy consumption patterns occuring every day and every week. This behavior cannot be observed in the minute data which also intuitively is sensible, as there is likely no repeating pattern within 2h. Strong repeating patterns in minute resolution data could potentially be observed in industrial energy consumers, as machines might run in regular intervals. The error curves for the sequence to sequence model are less steep, the higher the time resolution. This can be attributed to lower energy demand differences between single timesteps and less obvious patterns in the data. To further improve the forecasting accuracy both short and longterm, different measures could be undertaken. First of all, as Verdejo et al. (2017) do in their work, one could use weather forecasts and weather data in general which is of particular interest for hourly forecasts as a cold following day could mean increased heating is needed. Furthermore, the used dataset contains additional info on three separate room power measurements which could also be taken into account for more detailed info on individual behavior. In the work of Mocanu et al. (2016), they compute forecasts for each individual meter and which can then be aggregated to possibly create a more accurate entire household forecast. Another possibility is the combination of different resolution networks to use the hourly forecasts for 15 minute forecasts or the other way around. This could however be either benefitial or influence the networks negatively as predictions already have a relatively high error on any resolution. The LSTM could also be tuned and trained with more or less neurons. The current network seems to not perform as well as Marino et al. (2016), however it is not transparent how they computed their error metrics and it is also unknown how many epochs and which batch size was used by them. Then again, it is likely that LSTM is not the state-of-the-art for short sequence time series forecasting any more. Most current research on sequence to sequence forecasting stems from efforts in machine translation. However all methodology can always be transferred to time series forecasting as an embedded sentence is nothing else than a sequence of numbers. LSTMs, even with a sequence to sequence encoder decoder architecture, still have a number of limitations. Vaswani et al. (2017) report two important problems. First of all, LSTMs require sequential computation as each timestep depends on the previous computation. Therefore parallelizing the training workflow is very difficult. Secondly, information from previous timesteps is always handed through every cell and perturbed at every step. This means that at any given timestep, the network cannot directly refer to information from multiple steps in the past. Therefore they make use of *attention* as a new concept for sequence to sequence learning to consider important information from multiple time steps in the past. Attention was originally introduced by Bahdanau et al. (2014) and refers to the concept of considering the entire input sequence for every prediction step along with the previous predictions. However the input sequence is using learned weights such that the new prediction is impacted by parts of the sequence relevant for the current timestep. Other than attention as a concept, papers like Elbayad et al. (2018) show that convolutional neural networks can outperform LSTMs in sequence to sequence prediction using attention.

6 Conclusion

LSTM performs better than traditional methods to forecast energy demand and the S2S architecture further improves the network. The accurate single household long term forecasting can help both for planning power production for specific residential areas, as well as support energy trading such that long term purchase planning can be made. We did not achieve the impressive results of Marino et al. (2016) but can conclude that the proposed sequence to sequence architecure is both faster in forecasting and more accurate than the comparison architectures. However as errors are still around $0.6 - 0.8$ while all values for energy demand are between 0 and 4, it is important to see how well an aggregated forecast using single residential households compares to just forecasting the aggregated values of multiple households. On the other hand, there are numerous new techniques that outperform LSTMs in sequence to sequence learning tasks. A number of new algorithms and architectures should be used in an attempt to further improve residential household demand forecasting such that the total energy demand curve can be further improved. Yet this work is a good example for the benefits of employing deep learning for energy demand forecasting.

References

Aghaei, J. and Alizadeh, M.-I. (2013). Demand response in smart electricity grids equipped with renewable energy sources: A review. *Renewable and Sustainable Energy Reviews*, 18:64–72.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bonetto, R. and Rossi, M. (2017). Machine learning approaches to energy consumption forecasting in households. *arXiv preprint arXiv:1706.09648*.

Boulesteix, A.-L. and Schmid, M. (2014). Machine learning versus statistical modeling. *Biometrical Journal*, 56(4):588–593.

Depuru, S. S. S. R., Wang, L., and Devabhaktuni, V. (2011). Smart meters for power grid: Challenges, issues, advantages and status. *Renewable and sustainable energy reviews*, 15(6):2736–2742.

Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

Elbayad, M., Besacier, L., and Verbeek, J. (2018). Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*.

Ghofrani, M., Hassanzadeh, M., Etezadi-Amoli, M., and Fadali, M. S. (2011). Smart meter based short-term load forecasting for residential customers. In *North American Power Symposium (NAPS), 2011*, pages 1–5. IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. .

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hostick, D., Belzer, D., Hadley, S., Markel, T., Marnay, C., and Kintner-Meyer, M. (2012). Renewable electricity futures study. volume 3: End-use electricity demand. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States).

Huang, F. J., Boureau, Y.-L., LeCun, Y., et al. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.

Kiesel, R. and Paraschiv, F. (2017). Econometric analysis of 15-minute intraday electricity prices. *Energy Economics*, 64:77–90.

Kong, W., Dong, Z. Y., Jia, Y., Hill, D. J., Xu, Y., and Zhang, Y. (2017). Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid*.

Lee, K., Cha, Y., and Park, J. (1992). Short-term load forecasting using an artificial neural network. *IEEE Transactions on Power Systems*, 7(1):124–132.

Marino, D. L., Amarasinghe, K., and Manic, M. (2016). Building energy load forecasting using deep neural networks. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 7046–7051. IEEE.

Mocanu, E., Nguyen, P. H., Gibescu, M., and Kling, W. L. (2016). Deep learning for estimating building energy consumption. *Sustainable Energy, Grids and Networks*, 6:91–99.

Murphy, K. P. (2012). Machine learning: a probabilistic perspective.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

Siano, P. (2014). Demand response and smart grids—a survey. *Renewable and sustainable energy reviews*, 30:461–478.

Siegelmann, H. T. (1995). Computation beyond the turing limit. *Science*, 268(5210):545–548.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Verdejo, H., Awerkin, A., Becker, C., and Olguin, G. (2017). Statistic linear parametric techniques for residential electric energy demand forecasting. a review and an implementation to chile. *Renewable and Sustainable Energy Reviews*, 74:512–521.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.