Names: Narasimha Abhinav Koganti, Aliya Amirzhanova

Student numbers: 3445089, 3525113

Course: MSc Scientific Computing, MSc Scientific Computing

Email: koganti@stud.uni-heidelberg.de, amirzhanova@stud.uni-heidelberg.de

Date: 05.10.2018

Forecasting Energy Power Consumption

a report of mini research project

Advanced Machine Learning SS2018

Codes are available on github by: https://github.com/knabhinav/Machine-Learning

Contents

1 Introduction

Author: Aliya Amirzhanova

Forecasting plays an important and crucial part in many different fields such as data science, economics, finance, supply chain, marketing, weather and nature conditions etc. Having been prepared for the future helps to avoid unpleasant consequences and make right decisions.

Forecasting itself is the process of predicting future trends, values based on a collection of observations made sequentially through equally spaced periods of time [8]. Examples may include to predict future sales, prices or index trends, or to predict consumption rates.

Today's one of the main issues that is even on a state level is electricity consumption. In evidence, throughout the EU as one of the steps of energy policy goals by 2020, there is an increasing interest now in smart electricity solutions to achieve objectives in more intelligent control over electricity supply and consumption [13]. This can be done by smart electricity forecasting system, which is based on individual household consumption, allowing not only saving people's budget and encouraging them to use less electricity, but also might impact on a higher state level of electricity saving. If energy consumption and future projections are more transparent, then it would be easy to understand and control usage of it. Thus, to achieve leveraging metering solutions, proper methods, which can thoroughly study past events of the structural representation in time to develop a suitable model, are pivotal. A plenty of predictive methodologies are represented using classical statistical approaches including autoregressive models (AR), moving-average models (MA) and their different variations (ARMA, ARIMA, VAR, GARCH etc.) (out of the scope of this project). These methods are flexible in application and can capture stochastic processes by estimating parameters of the model. Another group can be considered Gaussian processes, which use a similarity measure between points to predict the values (out of the scope of this project, however, a Bayesian optimization for hyperparameters tuning in NN is used here).

On the other hand, main interest is now concentrated around deep learning techniques. Today there are proposed different types of Artificial Neural Networks (ANN) for modelling and forecasting of the time series. The most common and appropriate to the time series domain is the Recurrent Neural Networks (RNN) [15, 31, 39]. A Long

Short-Term Memory network (LSTM) and Gated Recurrent Unit network (GRU) are both from RNN family, were purposely built to address the vanishing and the exploding gradient problem [29], which is an issue for RNN. These networks have also an ability to efficiently learn long-range dependencies [29], which makes them efficient for time series analysis.

An alluring approach currently includes hybrid methods. These methods can combine several methods together into one model to assess the performance. Lots of them are based on combination of classical approaches with NN. There are recent hybrid methods using CNN+GRU technique, which were, however, applied on supervised classification task [18, 40] and obtained promising results. Inspired by this, we CNN+GRU technique on energy consumption time series to investigate its performance.

Therefore, this mini research projects aims to perform an enhanced approaches for predicting energy consumption at the household level, attempting short-term forecasting for a next hour ahead and for a day period ahead.

1.1 Objectives

The objective of this mini research project is to perform a comparative and thorough study of the following time series forecasting techniques:

- LSTM based neural network regression model.
- GRU based neural network regression model.
- CNN+GRU based neural network regression model.

The techniques are explored on the Electricity usage dataset (univariate data) from UCI Machine Learning repository. After appropriately modelling the above techniques on the dataset, the prediction / forecast values are obtained, which are then evaluated according the chosen loss metrics to assess and compare each model's performance.

1.2 Outline

This paper is structured in the following way. The next section describes data used for the project, outlining its characteristics and features. It also indicates data preprocessing techniques and making the data appropriate for the supervised learning. Later it gives a view on loss metrics used to assess the models' performances and compare between

them. The following section gives an overview on Recurrent neural networks prior to a detailed representation of the methods used in the project. Later, acquired results with their evaluation and discussion are illustrated. The last section concludes the report summarizing the project's insights, following by recommendation and future work.

2 Data description

Author: Narasimha Abhinav raKoganti

Individual household electric power consumption data set is taken from UCI Machine Learning Repository [17]. It consists measure of power consumed by a single household for a period of almost 4 years with one-minute sampling rate. The power consumption of a household varies based on various factors and has some trend and seasonality [3] (for example: It is high in winter, in leisure hours and weekends; and comparatively low during the business hours).

The dataset contains a total of 2075259 instances with string and float types. The values are measured between December 2006 to November 2010 (approximately 47 months).

No	Name	Description	Type of value
1	date	Date as dd/mm/yy format	String
2	time	time as hh:mm:ss format	String
3	global _active _power	Total power consumption in kilowatt	Float
4	global _reactive _power	household global minute-averaged reactive power	Float
5	voltage	minute-averaged voltage	Float
6	global _intensity	household global minute-averaged current intensity	Float
7	sub _metering _1	Power consumed in kitchen	Float
8	sub _metering _2	Power consumed in laundry room	Float
9	sub _metering _3	Power consumed by water-heater and AC	Float

The columns “voltage”, “global_active _power” and “global _intensity” in the dataset give the measurements of averaged per minute for the entire house respectively. Submetring1, Submetring2, Submetring3 are the measurements of the power usage in various sub-parts of a house. Global active power gives the entire power consumed in a house at the particular minute [17].

2.1 Cleaning data and filling missing values.

As we want to forecast only global _active _power, as the first step of data cleaning and preparation, we remove all other attributes than date, time and global _active _power.

The whole dataset contains 2570 rows (1.25%) of missing information. A few days, on which measurements for most of the minutes are missing, are removed, as it is hard to interpolate the missing values, and their removal does not really affect the quality of the model. The remaining missing data (the data missing only for a minute or two)

is filled with a “forward fill” function from pandas. A “forward fill” [1] is used, as the measurement of the next minute is not significantly different from the missing minute.

Data Preparation for the current project:
As we have enough data, it is more practical and useful to forecast for daily and hourly basis instead of every minute. Hence, two datasets are derived from the original dataset. The first dataset contains average active power consumed per each day, and the second one contains average active power consumed per each hour. Average active power per day is calculated by grouping and averaging per date. Models trained on this data are expected to predict/forecast daily average power consumption capturing the rise of power usage because of seasonal differences and the effect of weekends and festive days.

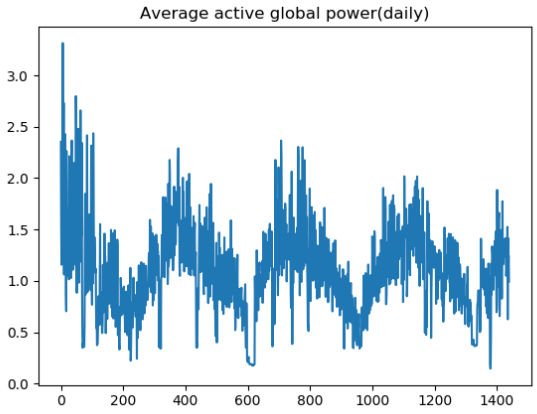

Figure 1: Averaged daily global power consumption.

Models trained on the second dataset, which contains average active power consumption per hour, are expected to capture the rise of power usage on mornings and evenings (leisure hours) and the trend of the rise in power usage on weekends.

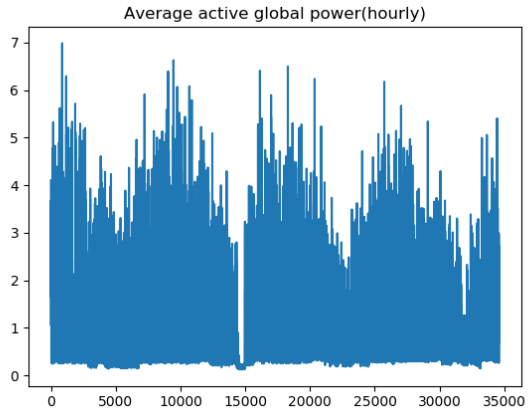

Figure 2: Averaged hourly global power consumption.

2.1.1 Scaling data:

Due to availability of enough data, training of neural networks is feasible to obtain promising results. We need to normalize the data for effective training of neural networks. One popular choice for time series is min-max scaling [21]. Min-max scaling: Min max scaling simply scales the given data to a range of [0,1].

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

2.1.2 Training and Test data:

After processing the data for two different datasets, we have in total of 1440 instances in the first dataset (daily average global active power) and 34560 instances in the second one (hourly average global active power). For daily average global active power dataset 1153 instances (80%) are used for training and 287 instances (20%) are used for validation. 27649 instances (80%) in hourly average global active power are used for training and 6911 (20%) are used for validation.

2.1.3 Loss metric

Loss functions help evaluating a model by comparison between predictions made by the model and the true values. As the objective of every model is to minimize the loss measure, thus using the right loss funtion is essential in obtaining good models. As we

use different models to explore and forecast the global active power, a common measure helps for making a comparison and selection between the models.

Here we choose two loss functions namely MAE (L1 Loss) and R2 _squared coefficients.

MAE/L1 Loss Mean Absolute Error (MAE) is the mean of absolute difference between the predictions and the true values. MAE works better on time series than MSE, as due to certain events at some random times the values in time series may fluctuate highly (outliers), and squaring the loss will result in very high/low error. Hence, MAE is more robust than MSE [37].

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{2}$$

R2 _squared coefficient

R2 _squared coefficient is also called coefficient of determination. It is the measure of how well the fitted regression line is close to the data points. However, R2 coefficient comes with a limitation such that a good R2 coefficient does not always really mean a good model. Thus, it is generally used along with the plots [22].

$$\begin{aligned} R^2 &= 1 - \frac{SS_{res}}{SS_{tot}} \\ SS_{res} &= \sum_i (y_i - f_i)^2 \\ SS_{tot} &= \sum_i (y_i - \bar{y})^2 \end{aligned} \tag{3}$$

2.1.4 Converting time series into supervised learning problem.

A typical machine learning model in supervised learning takes input(x) and predicts the output(y). However, time series data is single dimensional with only x values. Thus, we need to convert the data into the (input, output) format (make it as a supervised representation) [6], which can be then used for the models. This conversion into supervised learning problem is based on different factors. In the current setting for this problem at each time step we are going to forecast the Global active power for next day/hour. Hence, the input value(x) will be the global active power at a current day/hour and the output will be the global active power at the next day/hour. The table below shows (input, output) for first 10-time steps of the average daily global active power data.

Input	Output
2.35	1.53
1.53	1.15
1.15	1.54
1.54	1.19
1.19	1.62
1.62	3.31
3.31	1.77
1.77	1.90
1.90	2.73
2.73	-

3 Methods: theoretical background, implementation

Author: Aliya Amirzhanova

This section will provide an overview on RNN, their different configuration types, following by detailed LSTM, GRU and CNN+GRU methods used in the project. It also shows a technique on hyperparameters tuning and different optimization tools used with methods.

3.1 Recurrent Neural Networks

Author: Narasimha Abhinav Koganti

Traditional Artificial Neural Networks are mainly designed for classification and regression problems, where two different instances of input data are independent [21] of each other and there is no requirement to pass information across different instances. Thus, these Networks take a set of inputs and predict the corresponding outputs without passing any information to the next instance. Since the data in time series is not independent of each other, and values at each time step are dependent on one or more values of previous time steps; this makes traditional neural networks unfit to use with time series. Recurrent neural networks address the issue of dependency [34], so they are used widely for sequential data problems such as time series, music, speech recognition etc. A recurrent neural network shares information across time steps. At each time step the calculated activation is passed to the next time step.

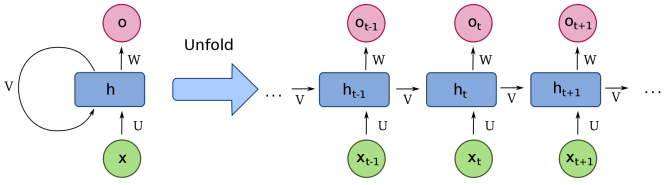

Figure 3: A general example of RNN [24]

Different types of RNN configuration

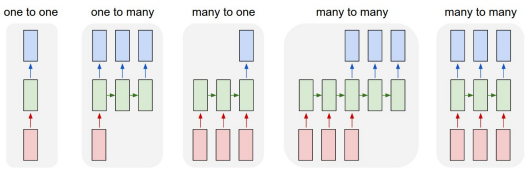

Figure 4: An example of different types of RNN configuration.

Many to many: For problems like machine translation the size of input data may vary from the size of output data, and at each time step the network needs to pass some information about the relation between words in input and output for the total output to be reasonable. Other application of many to many configurations include time series analysis, video classification etc.

One to many: Problems like image captioning take an image as input and produce a sentence as output. Some information needs to pass between output(words) at each time step for the output sentence to be meaningful.

Many to one: Problems like sentiment analysis take a sequence of words as input and predict the rating. Information needs to flow across each time step to identify the sentiment of the sentence.

Many to many for synced sequence input and output: Problems like video classification at each frame require continuous sequence of input and output.

In one to one. A one to one neural network is a vanilla neural network without any recurrence, it is generally used for classification and regression problems.

3.2 Basic functioning of an RNN

RNN equations

$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned} \tag{4}$$

where
h_t denotes the hidden state at time t.
X_t denotes the input at time t.
W_{hh} and W_{xh} denotes the network weight matrices to be trained.

3.2.1 Problems in recurrent neural networks

Vanishing gdradient problem.
During the training phase of an RNN the backpropagation is done after each epoch to adjust the weights. The weights receive an update proportional to partial derivative of

the error function [33]. When training is carried on a very long sequential data, due to the computation of partial derivative with chain rule, many small number (less than one) are multiplied depending on upon the depth of RNN, and, as a result, some gradients become too small and lead to inappropriate weight updates. This phenomenon is called vanishing gradient problem [33] . As a result, RNN fails to remember the information from time steps, which are far behind. However, the length between any two dependent time steps can be arbitrarily large in many sequential problems ,and the information from far behind time steps is crucial in the calculation of the current time step prediction [32]. Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) address this issue with the concept of cell state.

3.3 Long Short-Term Memory

LSTM networks have shown proven results and are well suited in the areas of speech recognition, time series etc. Like RNN the data is passed sequentially into an LSTM network. The fundamental difference between RNN and LSTM is in the units used at every time step, which is called LSTM cell.

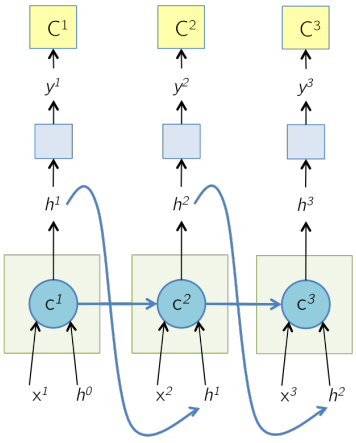

Figure 5: Forward pass in lstm diagram. [2]

LSTM architecture: A LSTM cell unit consists of a memory cell, activation gate, input gate, output gate and a forget gate. The memory cell in LSTM is a key part, which can store the required information over a long-range period. The forget gate, which is associated to the memory cell, decides which data to forget and which to remember,

and the memory is updated at each time step. As the memory cell can store values for a long range in LSTM, the effect of vanishing gradients is considerably very minimal when compared to RNN [20].

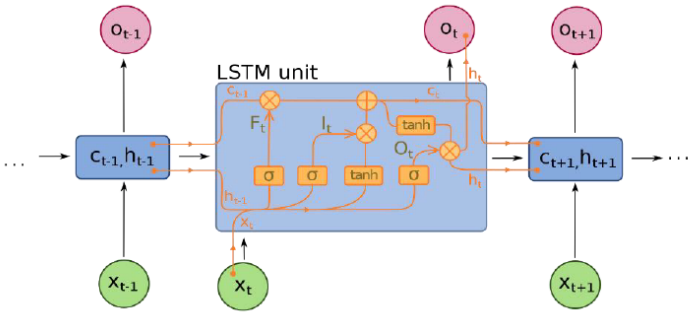

Figure 6: An example of LSTM cell. [9]

LSTM gates: Different gates operate together in a LSTM cell to produce a prediction. At each stage both input and a hidden state from the previous step are passed to all the gates and each gate has its set of own weights to operate on the input and a hidden state [11].

Activation gate: The activation gate operates on a hidden state and input to generate a vector of possible values for the current time step.

Input gate: The input gate also operates on a hidden state and input to generate vector for Hamdard product. This helps in deciding which values in vector generated by the activation gate are useful and the values obtained after performing the Hamdard product is carried forward to next gates.

Forget gate: Based on the hidden state from previous time step and input the forget decides which values from the memory are useful. It produces a vector for Hamdard product with a memory cell.

Output gate: Once the output from all the previous gates are combined in a specific format a vector of appropriate possibilities for the current time step are produced. The output gate choses from this vector based on the available hidden state information and input.

LSTM equations

$$
\begin{aligned}
a_t &= tanh(W_a \cdot x_t + U_a \cdot out_{t-1} + b_a) \\
i_t &= \sigma(W_i \cdot x_t + U_i \cdot out_{t-1} + b_i) \\
f_t &= \sigma(W_f \cdot x_t + U_f \cdot out_{t-1} + b_f) \\
o_t &= \sigma(W_o \cdot x_t + U_o \cdot out_{t-1} + b_o) \\
state_t &= f_t \odot state_{t-1} + a_t \odot i_t \\
out_t &= tanh(state_t) \odot o_t
\end{aligned}
\tag{5}
$$

Where out_t(hidden state) defines the ouput from lstm cell at time step t , $state_t$ define the the memory cell state at time step t,a_t, o_t, i_t, f_t denote the output of activation gate,input gate,forget gate and output gate respectively at time step t.

W_x, U_x, b_x with $x \in [a, i, f, o]$(for each gate).W and U contain weights of input and hidden state respectively and b is the bias.

Step by Step functioning in a LSTM cell. In a well-trained LSTM, where all the weights are adjusted by training with the available data, and the memory cell is ready for predicting the next time steps, the hidden state and input are passed to all the gates in LSTM cell simultaneously, then each gate generates its outputs, and later, these outputs from different gates are combined in a specific format to obtain the prediction.
At the first step the hidden state from the previous time step and the input is passed to activation gate, which generate a set of possible hidden features for the next time step. Then Hamdard product is performed between these features and the vector generated by the input gate. The intuition behind the Hamdard product is that each value in the vector generated by activation gate is multiplied with the probability of its occurrence for the next time step.

Simultaneously the forget gate also takes the hidden state and input to generate a vector of probabilities for the features stored in a memory cell. Then a Hamdard multiplication is carried between the probabilities vector and features stored in the memory cell. The intuition between this step is that its results produce the necessary features from the time steps far behind the current time step multiplied by its influence on the current time step. This is the main step, which gives LSTM edge over the traditional RNN.

Now an element wise addition is carried out between the results from the first step and the second step. The intuition behind this step is that now our set of collected features, after the element wise addition, is the combination of both, the features required from far behind steps and the recent time steps in required proportions.

Now the contents of the cell state are replaced by the collected possibilities to make the memory cell ready for time steps ahead. Then these collected possibilities are passed through a hyperbolic tangent funtion to avoid explosion of values, and Hamdard multiplication is done again between the values from output gate and the values obtained after passing through tangent function. The intuition behind this step is that the output gate gives weights to the collected features required for the current time steps.

The final output(features) Is then passed to a fully connected neural network, which predicts the values based on the features given from the LSTM cell. This final output is also called hidden state and is passed to the next time step [23].

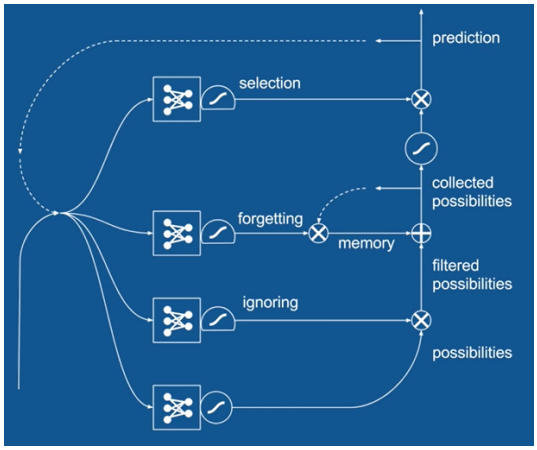

Figure 7: Workflow of LSTM cell.

3.3.1 Backpropagation with time

Training of RNN is carried using Backpropagation with time (BtPTT). BPTT starts with unfolding the recurrent neural network, and since every copy of network share the same set of parameters, BPTT tries to calculate the overall effect of the weights on the entire predicted sequence. The loss at every time step is carried backward all the way through intermediate steps till the first-time step (using chain rule) and the weights are updated accordingly [28] .

Back propagation in LSTM can be described by the following equations. [14].

$$
\begin{aligned}
\delta out_t &= \Delta_t + \Delta out_t \\
\delta state_t &= \delta out_t \odot o_t \odot (1 - tanh^2(state_t)) + \delta state_{t+1} \odot f_{t+1} \\
\delta a_t &= \delta state_t \odot i_t \odot (1 - a_t^2) \\
\delta i_t &= \delta state_t \odot \ a_t \odot (1 - i_t) \\
\delta f_t &= \delta state_t \odot state_{t-1} \odot f_t \odot (1 - f_t) \\
\delta o_t &= \delta out_t \odot tanh(state_t) \odot (1 - o_t) \\
\delta x_t &= W^T \cdot \delta gates_t \\
\Delta out_{t-1} &= U^T \cdot \delta gates_t
\end{aligned}
$$

Where

$$
gates_t = \begin{Bmatrix} a_t \\ i_t \\ f_t \\ o_t \end{Bmatrix} \qquad W = \begin{Bmatrix} W_a \\ W_i \\ w_f \\ W_o \end{Bmatrix} \qquad U = \begin{Bmatrix} U_a \\ U_i \\ U_f \\ U_o \end{Bmatrix} \tag{6}
$$

ΔT is the output difference as computed by any subsequent layer. Δout is the output difference as computed by the next time-step LSTM

The final update to internal prameters is given by

$$
\begin{aligned}
\delta W &= \sum_{t=0}^{T} \delta gates_t \otimes x_t \\
\delta U &= \sum_{t=0}^{T-1} \delta gates_{t+1} \otimes out_t \\
\delta b &= \sum_{t=0}^{T} \delta gates_{t+1}
\end{aligned} \tag{7}
$$

As we can see in the first equation Δout_t in back propagation represents the error passed from the losses accumulated at the time steps ahead.

Truncated Back Propagation through time : Nevertheless, the architectural advantage and the usage of the memory cell provides LSTM an edge over RNN, there is still exists a vanishing gradient problem. As the hidden memory cell is not infinitely large and cannot literally store all the values when trained on very long sequences LSTM still suffer from vanishing gradient. A popular and one of the effective approaches to solve this problem is Truncated back propagation through time (TBTT).

Truncated back propagation consists of two parameters k1 an k2. Where k1 is number of time steps shown to a network on a forward pass and k2 is the number of times to look at during a backward pass. k1 and k2 can be the same choice of their value depended on the data [28].

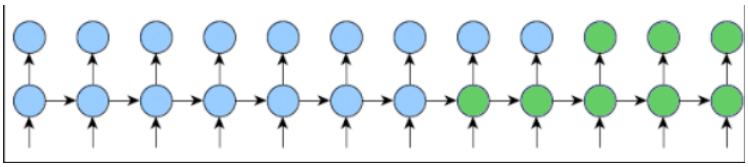

Figure 8: A diagram of truncated back propagation.

The above picture shows the diagram, where k1 $=$ 5 and k2 $=$ 3. Since truncated back propagation is carried after every few steps the propagation of a gradient is easier compared to normal BPTT, which helps in solving vanishing gradients problem.

Stacked LSTM. In traditional feed forward, neural networks it is proven that the depth of neural network attributes leads in success of a model. Stacked LSTM are inspired by the same idea and they have shown benchmark success in a field of speech recognition, time series etc. They can be interpreted as an abstraction similar to convolutions in normal neural networks, simultaneously passing the correlation with previous inputs [16].

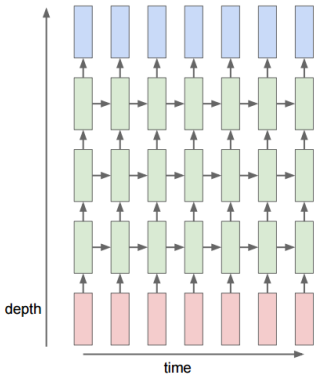

Figure 9: A diagram of stacked LSTM.

3.4 Finding hyper parameters

Finding the hyper parameter hidden state is important to derive an efficient model. However, since the training of a neural network is costly, it is clever to use an efficient search method instead of grid search or random search.

Bayesian optimization is a variant of a black box optimization and used in cases where the evaluation of objective is very expensive. It works with the help of surrogate model and an acquisition function [27].

Surrogate model: A surrogate model has the following properties:

- Surrogate model should be able to approximate the values of the true function
- Surrogate model should be cheap to evaluate.

Acquisition function: An acquisition funtion should estimate the profit of evaluating a point by using the surrogate model and it should balance between exploration and exploitation, for example, it should search in the regions, where the values are already expected to be high, as there is a greater chance to find another value which maximize the objective funtion. Simultaneously, it should also look in the areas with a high uncertainty, as there is chance to find new values, which maximize the objective function.

A Gaussian process acts as a good surrogate model (the details of this are out of scope for this report), and a popular choice for the Acquisition function is expected improvement [12].

Due to the lack of powerful GPU resources the Bayesian optimization for our case has been performed over a short range of values and a hidden state of size 123 is shown to give better results.

Final Architecture: LSTM (Input shape =1, Output shape = 123) → LSTM (Input shape =123, Output shape = 123) → Fully Connected layer (Output shape = 1).

3.5 Gated Recurrent Unit

Author: Aliya Amirzhanova

GRU stands for Gated Recurrent Unit, was firstly proposed by Cho et al. [10] in 2014. This type of network from the family of RNN is a variation of LSTM with fewer parameters resulting in combination of the forget and input gates into a single "update gate", and merging the cell state and hidden state [29]. Nevertheless, the performance of the GRU is comparable to the LSTM and can even outperform it. Moreover, less parameters of the GRU result in easy training of the model.

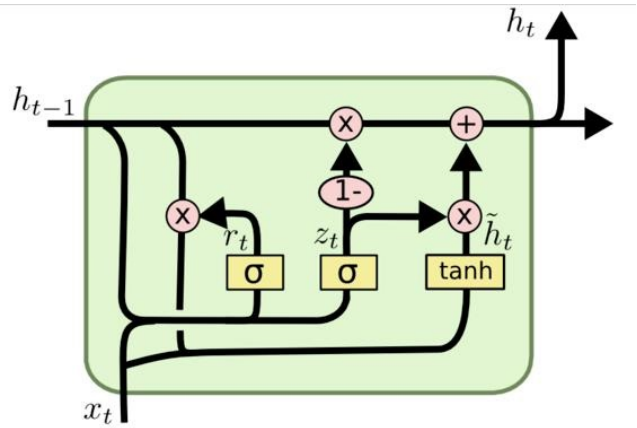

Figure 10: An example of GRU architecture [38].

$$
\begin{aligned}
z_t &= \sigma(x_t U^z + h_{t-1} W^z) \\
r_t &= \sigma(x_t U^r + h_{t-1} W^r) \\
\tilde{h_t} &= tanh(x_t U^h + (r_t \otimes h_{t-1}) W^h) \\
h_t &= (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h_t}
\end{aligned}
\tag{8}
$$

where r is a reset gate, and z is an update gate. The activation h_t at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h_t}$ [29]. The update gate controls and defines how much of the previous memory to keep, and the reset gate determines how to combine the new input with the previous memory [7]. Thus, compared to LSTM, GRU controls the information flow from the previous

activation, while LSTM lacks of this separate control. It is possible to obtain a vanilla RNN model by setting the reset gate to to all 1's and update gate to all 0's [7].

3.6 CNN + GRU

A convolutional neural network (CNNs) is a deep neural network that has been popular due to its successful performance in various classification tasks (for example, image recognition [26] or time series classification [36]). The CNN consists of a sequence of convolutional layers, the output of which is obtained by sliding the filter (kernel) over the input data, and at every location a computation of the dot product is performed. Thus, the model can learn filters, which are able to recognize specific patterns in the input data [4]. In case of time series forecasting the general idea is to study filters that represent certain repeating patterns in the series, and after getting the features use GRU on top to predict and forecast the future values. In fact, CNN might perform well on noisy data, due to their layered structure [4] by eliciting only relevant sequences.

3.7 Optimization algorithms

There exist different gradient descent optimization techniques to optimize neural networks. Gradient descent minimises an objective function by updating the parameters taking the steps proportional to the negative of the gradient of the objective function [30]. Step size is determined by the learning rate. Several of the optimization algorithms such as Stochastic gradient descent, Batch gradient descent, Mini-batch gradient descent deal with challenges such as getting trapped into a number of suboptimal local minima. Although, a real problem arises when reaching a plateu, which makes it tedious to escape, as the error derivatives for the hidden units will all become very tiny, even close to zero, and the error will not decrease [19].

Also, it is difficult to choose a suitable learning rate, to avoid painfully slowly convergence, and fluctuations around the minimum or even divergence [30].

A few optimization algorithms presented below are widely used, as they can overcome aforementioned issues. These different techniques have been implemented in this project.

RMSprop

Root Mean Square Propagation (RMSProp) adapts the step size separately for each weight based on the average of recent magnitudes of the gradients [19]. The algorithm works well on online and non-stationary tasks [5].

Adam

Adaptive Moment Estimation (Adam) computes adaptive learning rates for each parameter [25]. It is computationally efficient and suitable for non-stationary problems that are large in terms of data and/or parameters [25]. Adam uses the benefits of AdaGrad and RMSProp of storing an exponentially decaying average of past squared gradients [30], while also making use of the average of the second moments of the gradients (the uncentered variance) [25]. This allows it to behave "like a heavy ball with friction" [30], thus finding a flat minima.

L-BFGS

L-BFGS is an optimization algorithm in the family of quasi-Newton methods. L-BFGS uses an estimation to the inverse Hessian matrix, and unlike BFGS can store only retaining the most recent gradients, thus making it more efficient. However, as a result it might perform worse than BFGS.

4 Results and discussion

This section will give obtained results on each applied methods, on hourly and daily data.

4.1 Results for LSTM

Author: Narasimha Abhinav Koganti

The outputs obtained form the neural network have to be reverse normalized to calculate the forecast and prediction error.

Results for projecting on hourly data:

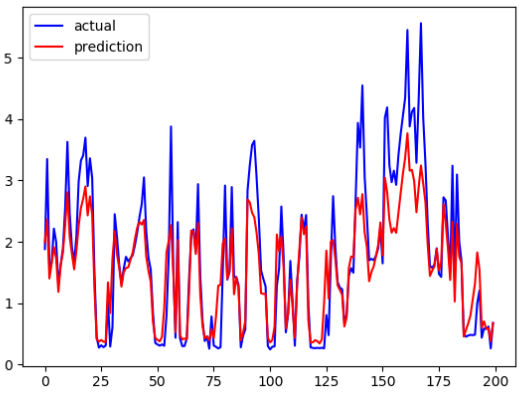

Figure 11: A resulting figure for prediction hourly.

When the network is back-propagated using normal BPTT it was unable to capture the seasonality in data properly but when back propagated with Truncated BPTT as we can see from the plot the model fits the data well. MAE loss of prediction is 0.412 after reverse normalizing the predictions.

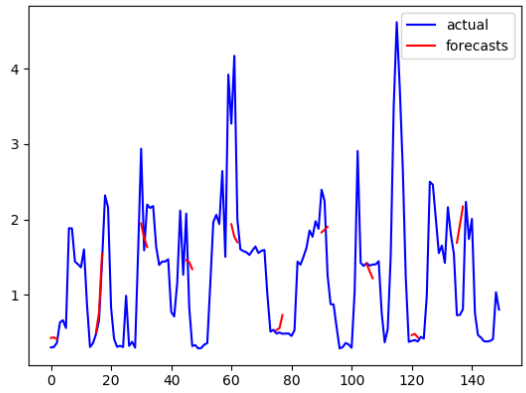

Figure 12: A resulting figure for forecasting hourly.

When forecasting using a trained LSTM network the input to the network should be the data starting form intial value till the point just before the start of forecast because at each time step hidden state and input is passed across the network and missing some initial values for input will change the hidden state and leads to errors in forecast. As we can see from the plot the forecasts are close to the true values. MAE loss of short-term forecasts is 0.48 on reverse normalized forecasts.

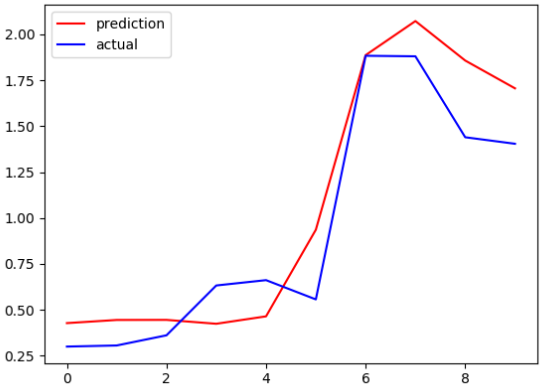

Figure 13: A resulting figure for forecasting long-term.

As expected when the LSTM is used to forecast over long range values the forecast goes bad because the small errors in previous time steps are accumulated. MAE loss for long range forecasts is 1.47 on reverse normalized data.

Results for projecting on daily data:

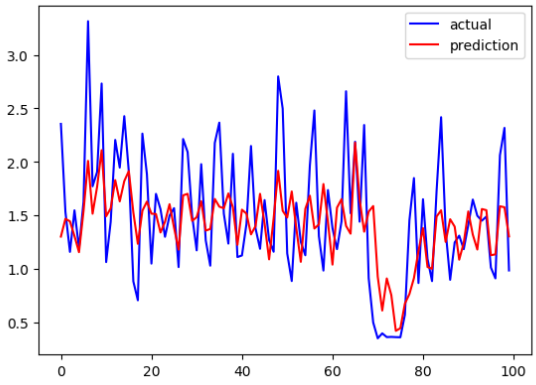

Figure 14: A resulting figure for prediction daily.

The prediction looks good with a mae loss of 0.22.

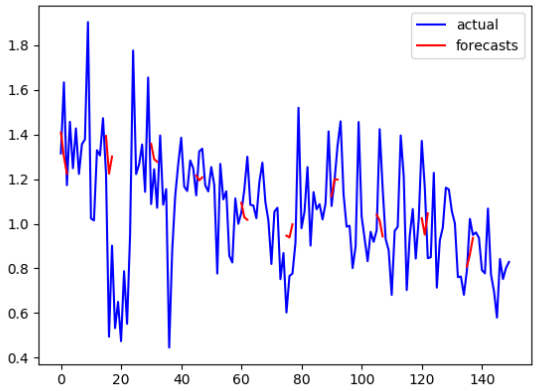

Figure 15: A resulting figure for forecasting daily.

As we can see from the plot the forecasts are close to the true values. MAE loss of short-term forecasts is 0.1889.

As expected, when the LSTM is used to forecast over long range values it is able to capture rise in power during leisure to a major extent. MAE loss for long range forecasts is 1.04.

4.2 Discussion for LSTM

Author: Narasimha Abhinav Koganti

The forecasting of global active power for both daily and hourly has been performed with different models such as LSTM, GRU and CNN+GRU along with different number of stacked layers, various hyper parameters and multiple variants of back propagation. Among all the model constructed a two-layer stacked LSTM with truncated back propagation of 150 steps (k1 = k2 = 150) seems to fit the data well and was less prone to errors on forecasting.

4.3 Results and discussion for GRU

Author: Aliya Amirzhanova

In order to obtain the best fitting model different parameters have been searched, comparing to each other. Thus a number of hidden layers were selected between [16, 51, 64, 128, 264, 512]. As the number was increasing the network was becoming dense and the training time was substantially increasing from around 10mins to 10-12hrs for dense models. Surely, the dropout technique can be used here, however, it is a method used to avoid overfitting, while output of this dataset was always too good on test data rather on train. Hence, promising results are obtained with hidden layers of [51, 64, 128], and among them using 64 has better performance. Also batch size was used to be between [1, 16, 128, 256]. The batch size of 1 produces adequite results compared to others, however the best batch size is 16. After analysing the loss of NN and also visually, it has been decided for epoch size to be 400. One more interesting technique, which has been tried with this data is sequence length. This technique allows using different types of configuration described earlier. For GRU it is used single input - single output, which is appropriate for time series, also many to one is used. Thus, sequence length has been tried to use [3, 6, 10]. However, using sequnce length is hard to NN to learn, and thus it produces poor results. Hence using sequence length of 1 is the most approprite solution to fit the model to this data. Moreover, different learning rates was used for comparison.

Results for projecting on hourly data:

The figure below shows an output with found best hyperparameters to be 64 hidden layers, batch size of 16 and learning rate of Adam 1e-4. The training epoch size is 400. R2 score is 0.9386 and MAE is 0.1117 on inverted back data.

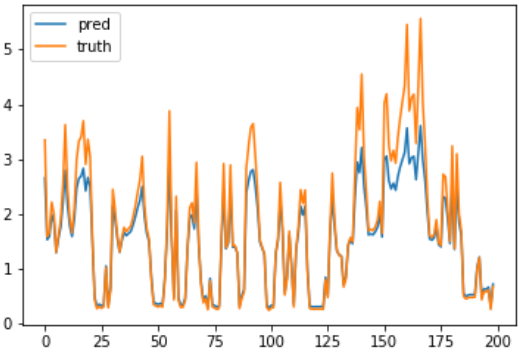

Figure 16: A figure of GRU prediction hourly.

The figure (fig. 17) shows result of using hidden dimension of 256 layers and it indicates that using 64 layers performs better and fits more precise than using 256.

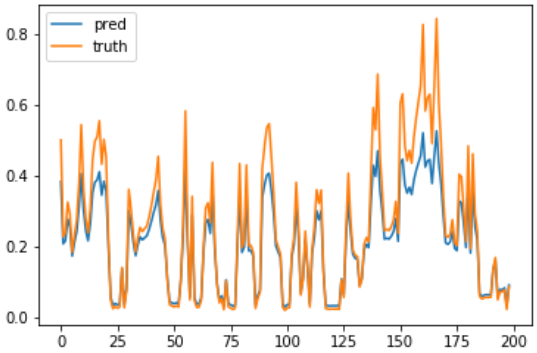

Figure 17: A figure of GRU prediction hourly.

Thus, the figure below shows a result of t+1 forecasting on the test set based on the best acquired fitting model. R2 score is 0.9635 and MAE is 0.0757 on inverted back data.

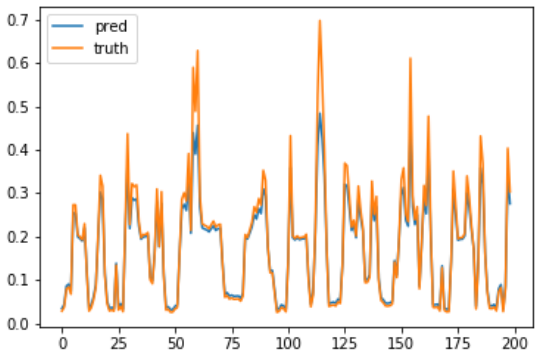

Figure 18: A figure of GRU forecasting hourly on test set.

As it was mentioned, using sequence length does not fit well even on train set (fig. 19), thus this type of technique is not appropriate for forecasting especially on this dataset. This figure below shows a result of using sequence length of 3, as the length increase the output becomes more deficient.

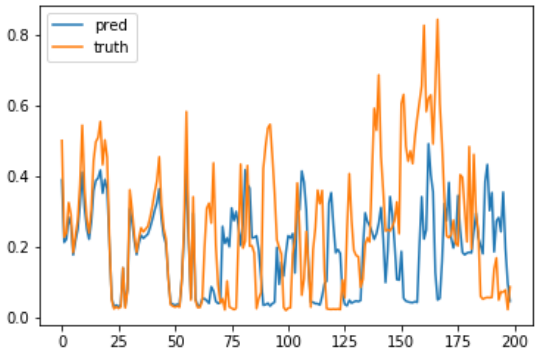

Figure 19: A figure of GRU forecasting hourly with sequences.

Results for projecting on daily data:

On daily data the same hyperparameters were selected, and batch size equal to 16 performs better with R2score equal 0.791 on train and MAE of 0.1148.

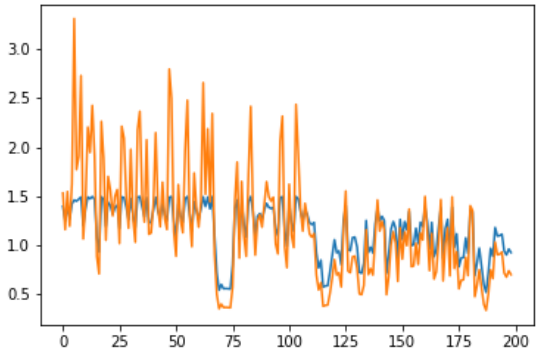

Figure 20: A figure of GRU prediction daily.

R2score equal to 0.923 on test and MAE of 0.064.

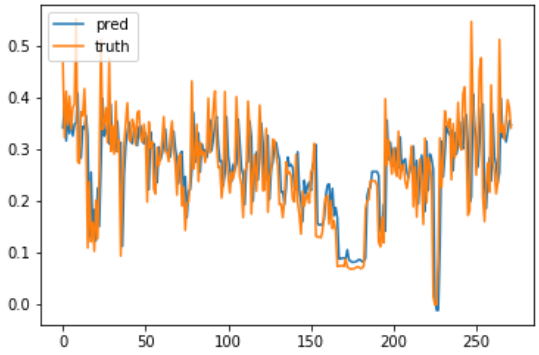

Figure 21: A figure of GRU forecasting daily.

4.4 Results and discussion for CNN+GRU

For this type of model Conv1d was used with BatchNorm1d, kerner size has been chosen to be 3 and padding 2. At first CNN was trained on the data, so that it would be possible to learn the data features, and later on pre-trained CNN, GRU has been applied, so that learning rate of CNN is relatively and considerable small compared to GRU. Initially, it has been considered that this model might perform better than others with learned features due to CNN, however, this model works worse than GRU itself. Probably other hyperparameters of kernel size and padding and hidden dimension might make the model better, however, possibly the specific residual component of the time series is difficult to capture.

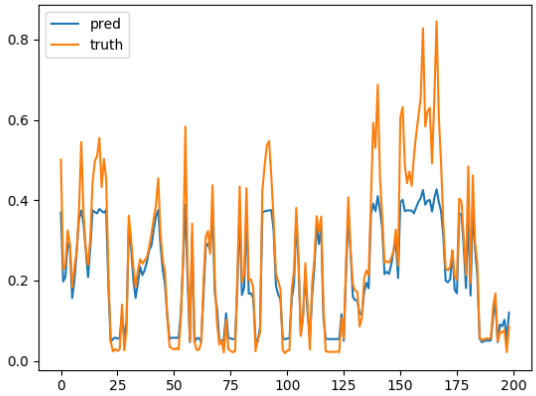

Figure 22: A figure of CNN+GRU forecasting daily.

Anyway R2score is considerably high being 0.8283 on train and 0.9034 on test set. MAE is 0.143 on train, and on test.0.132.

4.5 Overall comparison

Author: Aliya Amirzhanova

Thus, it can be seen that the best model to fit this data and predict t+1 is GRU, as it produces decent results with tuned hyperparameters. In addition it has been stated previously that GRU might outperform LSTM even if they are simpler than LSTM in structure. However, probably their ability to control information flow from the previous activation might help to understand the data more in depth and learn its structure. However, it does not help much for long-term forecasting such as t+3. As the forecasted output mostly goes up, despite the original curve. However, it is still possible then for long-term to have not the forecasted value, as it will be not as expected, but a trend of the line, forecasting either it will go up or down (as it might capture the up/down behaviour). In addition, probably even for this it is better to have a hybrif model either with classical approaches or with SVC / SVR (as they also show strong performance). Despite this, for short-term prediction of t+1 all models actually performs well, and GRU can be considered as better smart solution for this type of data.

Also the results have been performed using different optimization techniques such as Adam, RMSprop, L-BFGS. L-BFGS takes long computaional times, and compared with the others concedes to Adam and RMSprop. Adam and RMSprop behave slightly similar,

as their difference in output performance is not much. However, Adam is in a leading position. In evidence, as it is shown in [25] Adam works well in practice and can "outperform RMSprop towards the end of optimization asradients become sparser" [25].

5 Conclusion

Author: Aliya Amirzhanova

To sum up, this project work has been aimed to apply and compare different deep learning techniques of LSTM, GRU, CNN+GRU to predict / forecast of energy consumption on hourly / daily basis on the household level. These methods were done using different techniques and configurations. Also three various optimization algorithms were applied. In addition Bayesian optimisation technique is used for hyperparameters tuning. From the presented results it can be clearly seen that applied NN techniques can capture the trend, and make predictions, however, at forecasting, especially on long-term period their performance is not very accurate and encouraging. Therefore, below several recommendations for further work are given.

5.1 Recommendation and further work

Author: Narasimha Abhinav Koganti

Encoder-decoder architecture: An Encoder-Decoder architecture is a popular approach used in machine translation. The intuition of this architecture is that the encoder derives hidden feature while the decoder takes them as input and tries to predict the values based on the derived features.

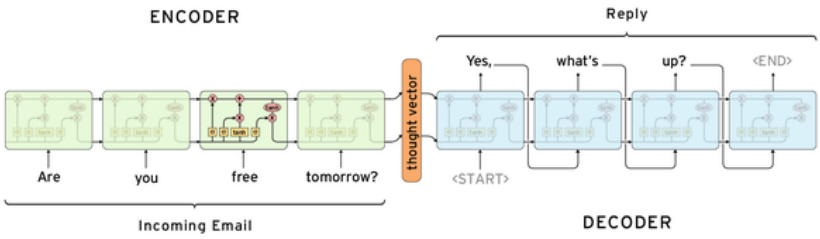

Figure 23: An example of encoder-decoder architecture.

Also, Bayesian optimization can be performed on a large range of values with the use of powerful GPU.

Author: Aliya Amirzhanova

Perhaps, another recommendation for further work might be a combination of used methods with classical statistical approaches. The time series is considered to be structured from two additive components: the interpretable component i_t and the residual

component e_t: $y_t = i_t + e_t$ [35]. Thus, while keeping the interpretability of the model with classical statistical approach, [35].

In addition, using mutivariate time series, obtaining data-features that might affect the energy consumption (in our case global active power) and analyisng it together with the main data might improve the results.

A different way might be using a Fourier transform to investigate the series in the frequency domain and later apply NN to learn and produce forecasting in this domain.

References

1. URL: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html.
2. arunmallya. "LSTM Forward and Backward Pass". URL: http://arunmallya.github.io/writeups/nn/lstm/index.html/.
3. BenAnderson, SharonLin, AndyNewing, AbuBakrBahaj, and PatrickJamesa. "Electricity consumption and household characteristics". *ScienceDirect*, 2016.
4. A. Borovykh, S. Bohte, and C. W. Oosterlee. "Conditional time series forecasting with convolutional neural networks." *arXiv:1703.04691v4 [stat.ML]*, 2018.
5. J. Brownlee. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.
6. J. Brownlee. "How to Convert a Time Series to a Supervised Learning Problem in Python". URL: http://blog.minitab.com/blog/adventures-instatistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit.
7. I. Changhau. "LSTM and GRU – Formula Summary", 2017. URL: https://isaacchanghau.github.io/post/lstm-gru-formula/.
8. C. Chatfield. *Time Series Forecasting*. Chapman & Hall/CRC, 2000.
9. G. Chevalier. "Long short-term memory". URL: https://en.wikipedia.org/wiki/Long_short-term_memory.
10. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". *arXiv:1406.1078 [cs.CL]*, 2014.
11. Colah. "Understanding LSTM Networks". URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
12. P. I. Frazier. "A Tutorial on Bayesian Optimization". *arXiv:1807.02811*, 2018.
13. K. Gajowniczek and T. Ząbkowski. "Electricity forecasting on the individual household level enhanced based on activity patterns". *PLoS ONE* 12:4, 2017. DOI: 10.1371/journal.pone.0174098.
14. A. Gomez. "Backpropogating an LSTM: A Numerical Example". URL: https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9.

15. A. Graves. “Generating sequences with recurrent neural networks.” *arXiv:1308.0850*. 2013.

16. e. a. Graves. “Speech Recognition with Deep Recurrent Neural Networks, Neural and Evolutionary Computing”. *<https://arxiv.org/pdf/1303.5778.pdf>*, 2013.

17. G. Habrail and A. Barard. “Individual household electric power consumption Data Set”. URL: https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption.

18. H. Harutyunyan and H. Khachatrian. “Combining CNN and RNN for spoken language identification”, 2016. URL: https://yerevann.github.io/2016/06/26/combining-cnn-and-rnn-for-spoken-language-identification/.

19. G. Hinton. “Neural Networks for Machine Learning”, n.d. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

20. S. Hochreiter and J. Schmidhuber. “Long short-term memory”. *Neural Computation*, 1997.

21. B. IA and H. M. “Artificial neural networks: fundamentals, computing, design, and application”. *Journal for Microbiolgical Methods,* 2000.

22. F. J. “Regression Analysis”. URL: http://blog.minitab.com/blog/adventures-instatistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit.

23. E. Kang. “Long Short-Term Memory (LSTM): Concept”. URL: https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359.

24. A. Karpathy. “The Unreasonable Effectiveness of Recurrent Neural Networks”. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

25. D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. *arXiv:1412.6980 [cs.LG]*, 2014.

26. A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” *Advances in Neural Information Processing Systems* 25, 2012, pp. 1097–1105.

27. J. Mockus. “Bayesian approach to global optimization: theory and applications”. *Kluwer Academic*, 2013.

28. M. C. Mozer. “A Field Guide to Dynamical Recurrent Neural Networks.” *IEEE Press, Hillsdale, NJ: Lawrence Erlbaum Associates*, 1995.

29. R. Rana, J. Epps, R. Jurdak, N. Lane, X. Li, R. Goecke, M. Breretonk, and J. Soar. "Gated Recurrent Unit (GRU) for Emotion Classification from Noisy Speech". *arXiv: 1612.07778v1*, 2016.

30. S. Ruder. "An overview of gradient descent optimization algorithms", 2016. URL: http://ruder.io/optimizing-gradient-descent/.

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

32. L. Sutskever, O. Vinyals, and Q. Le. "On the difficulty of training Recurrent Neural Networks". *arXiv:1211.5063*, 2012.

33. O. Sutskever L.and Vinyals and Q. Le. "On the difficulty of training Recurrent Neural Networks". *arXiv:1211.5063*, 2012.

34. O. Sutskever L.and Vinyals and Q. Le. "Sequence to Sequence Learning with Neural Networks". *Electronic Proceedings of the Neural Information Processing Systems Conference.* 2014.

35. B. A. Swastanto. "Gaussian Process Regression for Long-Term Time Series Forecasting". *Master Thesis, TU Delft*, 2016.

36. Z. Wang, W. Yan, and T. Oatesn. "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline." *CoRR, abs/1611.06455*, 2016.

37. C. Willmott and K. Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". *Climate Research, 30, 79–82. DOI:10.3354/cr030079*, 2005.

38. J. Xinyang. "GRU network for deep learning", n.d. URL: https://www.codeblogbt.com/archives/458479.

39. W. Zaremba, I. Sutskever, and O. Vinyals. "Recurrent neural network regularization." *arXiv:1409.2329*, 2014.

40. Z. Zhang and L. Luo. "Hate Speech Detection: A Solved Problem? The Challenging Case of Long Tail on Twitter." *arXiv:1803.03662 [cs.CL]*, 2018.