Flexible smart metering for multiple energy vectors with active prosumers

Project Duration: 2015-01-01 – 2017-12-31
Type: CP

<table>
<thead>
<tr>
<th>WP no.</th>
<th>Deliverable no.</th>
<th>Lead participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP4</td>
<td>D4.5</td>
<td>TI</td>
</tr>
</tbody>
</table>

Report on Load Generation Forecasting

Prepared by: Ennio Grasso (TI)
Issued by: FLEXMETER Project Office
Document Number/Rev.: FLEXMETER/D4.5/V3.0
Classification: Public
Submission Date: 2016-11-23
Due Date: 2016-10-31

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 646568

©Copyright 2015 POLITECNICO DI TORINO, IREN ENERGIA SPA, STMICROELECTRONICS SRL, TELECOM ITALIA, RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN, INSTITUT POLYTECHNIQUE DE GRENOBLE, UNIVERSITATEA POLITEHNICA DIN BUCURESTI, SIVECO ROMANIA SA, ALMA MATER STUDIORUM – UNIVERSITA’ DI BOLOGNA, E-ON SVERIGE AB. This document and the information contained herein may not be copied, used or disclosed in whole or in part outside of the consortium except with prior written permission of the partners listed above.
### Document

<table>
<thead>
<tr>
<th>Title</th>
<th>Press Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Deliverable</td>
</tr>
<tr>
<td>Ref</td>
<td>D4.5</td>
</tr>
<tr>
<td>Target version</td>
<td>V3.0</td>
</tr>
<tr>
<td>Current issue</td>
<td>V3.0</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>File</td>
<td>D4.5_flexmeter_v3.0.pdf</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Ennio Grasso (TIM), Vinh Le-Van (INPG), Julien Sandraz (INPG), Yvon Besanger (INPG), Luca Barbierato (POLITO)</td>
</tr>
<tr>
<td>Reviewer(s)</td>
<td>All consortium, Staffan Hjort (EON)</td>
</tr>
<tr>
<td>Apprver(s)</td>
<td>Andrea Acquaviva</td>
</tr>
<tr>
<td>Approval date</td>
<td>2016-11-23</td>
</tr>
<tr>
<td>Release date</td>
<td>2016-11-23</td>
</tr>
</tbody>
</table>

### Distribution of the release

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>PU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution list</td>
<td>All Consortium</td>
</tr>
</tbody>
</table>

### History of Changes

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/09/2016</td>
<td>V0.0</td>
<td>Table of Content</td>
</tr>
<tr>
<td>15/09/2016</td>
<td>V1.0</td>
<td>Skeleton ready</td>
</tr>
<tr>
<td>20/09/2016</td>
<td>V1.0</td>
<td>Agreed work split</td>
</tr>
<tr>
<td>15/10/2016</td>
<td>V2.0</td>
<td>Partners contributions ready</td>
</tr>
<tr>
<td>20/11/2016</td>
<td>V3.0</td>
<td>Integration and harmonization</td>
</tr>
<tr>
<td>23/11/2016</td>
<td>V3.0</td>
<td>Final Delivery</td>
</tr>
</tbody>
</table>
# Contents

Document .................................................................................................................. 2  
Distribution of the release ...................................................................................... 2  
History of Changes ................................................................................................. 2  
Contents .................................................................................................................. 3  
1 Executive Summary ............................................................................................ 4  
2 Introduction ......................................................................................................... 5  
3 Load forecasting ................................................................................................ 7  
   3.1 Introduction .................................................................................................... 7  
   3.2 Short term load forecasting methods .............................................................. 7  
      3.2.1 Brief literature review ............................................................................ 7  
      3.2.2 Artificial Neural Network models ....................................................... 8  
   3.3 Multi-Layer Perceptions and ANN models design ......................................... 11  
      3.3.1 Multi-Layer Perceptions and training process .................................... 11  
      3.3.2 ANN models design ............................................................................ 13  
   3.4 Application and numerical results ................................................................. 16  
   3.5 Conclusion .................................................................................................... 18  
4 Distributed Generation forecasting .................................................................... 20  
   4.1 Analysis of predictor variables - the weather data ....................................... 20  
   4.2 How to exploit past and future data .............................................................. 21  
   4.3 seq2seq for irradiance forecast ................................................................... 24  
   4.4 General architecture .................................................................................... 24  
   4.5 Dense embedding of discrete variables ...................................................... 25  
   4.6 What to encode and what to decode ............................................................. 26  
   4.7 Some implementation details and results .................................................... 27  
5 Conclusions ....................................................................................................... 29  
Bibliography .......................................................................................................... 30
1 Executive Summary

Today, the growing penetration of renewable energy source in distribution network, especially solar and wind power, presents to electric utilities operational system challenges and poses issues for the utility in matching load with generation. In the residential sector, the integration of PV system on buildings offers operational flexibility to distribution system operators and possibility to reduce the bill to prosummers. The realization of these promising possibilities requires developing a load and generation forecasting (LGF) tool. In general, the flexibility on the demand side could be used by the Energy Aggregators to optimize their portfolio as well as by system operators to balance the power. In FLEXMETER project, an Energy Aggregators Platform has been developed and LGF is one component in this platform.

Overall, this document focuses on the development of residential load and PV system production forecast during task T4.5 activities. The task has been subdivided into two independent parts, load forecast and generation forecast, which are performed by two partners INPG and TIM. As a component of the Energy Aggregators Platform, these works have a strong collaboration with task T4.3.
2 Introduction

Flexmeter Platform is a cloud-based smart multi metering infrastructure. The proposed architecture is in charge to manage smart meters of different utilities (e.g. electricity, water, heating and gas) providing features to register new devices, to enable a real-time bidirectional communication with deployed smart meters along the distribution network, to aggregate and correlate such energy related information, to perform (near) real-time data processing and to provide a set of API and tools to access devices and post-processed information. Summarizing all the functionalities, the main purpose of this smart metering architecture is enabling and fostering new services to make a more reliable and secure grid network management. In Figure 1, it is presented an high level overview of the Flexmeter Platform. It is a three layer architecture with:

- a Device Integration Layer

Flexmeter Platform deal with heterogeneous devices that exploit different communication protocols (e.g. IEEE 802.11, ZigBee, SCADA or 6LowPan). The Device Integration Layer is the lower layer of the proposed infrastructure. It abstracts different hardware data sources leveraging upon entities called Device Integration Adapters (DIA). A DIA converts the measurements coming from the devices to the Flexmeter Data Format and send them to the cloud infrastructure via MQTT, which exploits publish/subscribe communication paradigm. In addition, we developed a specific DIA to integrate Real-Time Digital Simulators, such as RTDS and Opal RT, into the Flexmeter infrastructure.

- a Middleware Layer

The Multitenant IoT Platform Layer is composed by several software components acting together to allow MQTT communication with Device Integration Adapters, to receive, control and store measurement messages and to provide REST APIs to access data, devices, assets and maintenance operations. The Message Broker provides an asynchronous communication through MQTT. MQTT is a publish/subscribe protocol to send data in (near-)real-time. This approach decouples network traffic between information producers and consumers increasing the scalability of the whole infrastructure. The Message Broker is also in charge of routing all events to the Communication Engine. In its core, the Communication Engine consists of other two blocks: i) Event Sources and Inbound Pipeline ii) Commands Destination and Outbound Pipeline. The Event Source receives messages from the Message Broker, checks the integrity and push them into the Inbound Pipeline. Inbound Pipeline manages message network traffic spikes to relieve the database interface from congestion and to ensure the measurements storage. Instead, the Outbound Pipeline receives command requests from the REST API interface or from internal routine, manage command traffic spikes to ensure command delivery and push them into the right Command Destination. Command Destination encapsulate the command in JSON format, choose the right topic to reach the selected device and send the message to the Message Broker. Data Storage is an interface designed to integrate different non relational database models (e.g. document oriented or time series). This approach helps in scalability and clusterization keeping independent the cloud infrastructure from the low-level database management systems. Device Management handles the interactions between devices and application layer. Asset Modules manages different information regarding people, places and things that are called assets in Flexmeter. The main asset represents physical location of electric smart meters. It contains all of the core asset management calls including CRUD methods for asset categories and assets. Finally, REST API Interface Manager provides REST API to access information and manage entities or devices in the infrastructure. All REST calls are subject to authentication to check if the user is authorized in performing operations.
• an Application Layer

The Application Layer offers the interface for different applications to design complex services.

To better understand the position in Flexmeter Platform of the Load and Generation Forecasting algorithms that are going to be presented in the following chapter, we draw a red dot circle around the services in Figure 1 that transform the algorithm into accessible energy management services.

Figure 1 - Flexmeter Platform and Load and Generation Forecasting algorithms encapsulation
3 Load forecasting

3.1 Introduction

Electric load forecasting is a topic of great interest for electric utilities and system operators. Reliable operations as well as pacification of the power system require load forecasting. Based on the lead times, it can be divided into four major categories:

- very short-term load forecasts (VSTLF) for single minute to several dozen minutes ahead, used for the load-frequency control functions in Energy Management System. Due to some measurement delays, the VSLTF is often replaced by short-term load forecasts to fulfil the need in network operations.
- short-term load forecasts (STLF) for one hour to one week ahead, used for advanced distribution automation function in Distribution System Operators (DSO), supplying information for system management of day-to-day operations and for the retail market;
- medium-term load forecast (MTLF) for one week to a year, used for the purpose of scheduling fuel supplies, negotiation of electricity contracts, and unit maintenance;
- long-term electric load forecast (LTLF) for more than one year, used to supply electric utility company management with prediction of future needs for expansion, equipment purchases.

Load forecasting is an extensively investigated subject on the transmission level, high and medium voltage (HV & MV) [N. Ding et al, 2016]. However, on the distribution level, few works have been proposed. This can be explained by two principal reasons: first, more attention has been paid to the transmission level, as the transmission grid, involving huge costs, is the backbone of the power systems. Second, there were not available measurement on low voltage level (LV), relaying on which the forecasting models can be designed and validated. Due to the expansion of the smart grids in recent years, load forecasting on low voltage level receives more attention from research communities, electric utilities as well as system operators. For innovative services for the DSOs and prosumers proposed in FLEXMETER project, Grenoble INP – G2Elab's contribution focuses on the STLF on the distribution network. The scope of our work is to develop a STLF models design based on the artificial neural network.

The rest of this chapter is organized as follow. First, we start by Section 2.2 with a brief reviewing of different models found in literature, their specificities and applications to the short term load forecast. Then, developed methodologies and algorithm for ANN model design in FLEXMETER project are presented in Section 2.3, followed some results of testing the algorithm in Section 2.4. The Section 2.5 proposes technique for integration of the load forecasting module into the Energy Aggregator Platform. Finally, we conclude the chapter in the Section 2.6.

3.2 Short term load forecasting methods

3.2.1 Brief literature review

A wide range of methodologies and model for forecasting are presented in the literature. They may be classified into three broad categories: parametric, nonparametric, and artificial intelligence based methods. The parametric methods are based on the assumption that there is an existence of internal structure within data.
Their principle is to formulate a mathematical or statistical model of load by examining qualitative relationships between the load and exogenous factors that influence to the load, especially weather, calendar and social variables.

Some examples of this category are Auto Regressive, Moving Average, Auto Regressive Moving Average (ARMA), etc. Nonparametric methods allow a load forecast to be calculated directly from historical data. Unlike parametric models, nonparametric models do not require the modeller to make any assumptions about the relationship between load data and exogenous variables. For example, the similar day approach mainly relies on the choices of the comparison characteristics between past observations and the forecasting day.

Both parametric and nonparametric methods refer to classical approach. They are attractive because some physical interpretation may be attached to their components. However, they are basically linear devices, and the load series are known to be distinctly nonlinear functions of the exogenous variables [H. S. Hippert et al., 2001].

The application of artificial intelligence (AI) technologies to the load forecasting is becoming more and more widespread nowadays, mainly because of their flexibility, explanation capabilities and teaming nonlinear relationship between influence factors and historical load data. The AI-based methods include expert systems, fuzzy logic, and artificial neural networks (ANN). These methods have been tried out [N Ding et al. 2016] and the ANN are undoubtedly the one receiving the largest share of attention [H. S. Hippert et al., 2001].

In distribution systems, many works show considerable success in forecasting accuracy of distribution loads by using ANN. However, the justification of the choices for the input vector is often absent [A. Capasso et al., 1994], [R. Lamedica et al., 1996], [A. Khotanzad et al., 1998], [G. Dreyfus et al., 2001], and the model architecture is often not systematically argued or commented in a clarified way [N Ding et al. 2016]. In FLEXMETER architecture, we design neural network models for short term load forecasting. Real data collected from smart meter systems installed in two demonstrator sites (Malmo and Turin) are used in order to validate proposed models.

3.2.2 Artificial Neural Network models

Over the past two decades, the ANN in the AI-based methods has been applied in various fields, e.g., pattern recognition, robotics, and load forecast, to name a few. Thanks to its non-linearity and great learning ability, ANN quickly gained its popularity in load forecast. It is widely used in all forecasting lead times and all electrical voltage hierarchies. Compared to the classical approaches, it is often considered as a benchmark of performance [J.W. Taylor, 2010], [J.M.C. Sousa, 2009] or more often as a complimentary to enhance their abilities facing dynamic varying environments.

The use of the ANN is often divided into two phases: a learning phase and a prediction phase. In the learning phase (also known as training phase), ANN adapts its parameters, namely the synaptic weights and biases, to learn the knowledge within inputs and outputs containing in the historical data set. In the prediction phase, parameterized ANN is used as a transfer function to produce the output(s). ANN is very flexible as when the operating environment changes, with the same structure, it is capable of rapidly adapting its interconnection weights to respond to the changing situation after the training process. ANN is also known to have great skills coping with the noisy data set [J.M.C. Sousa, 2009].

The disadvantages are related to its large scale choices, such as for the architecture (feed forward or recurrent, etc.), for the structure (number of hidden layers and number of hidden neurons on each layer), for the learning set, and the number of interactions for the training (to avoid overflow). The justification of the choices is not systematically argued or commented in a clarified way [J.M.C. Sousa, 2009]. The ANN is
considered as a class of algorithms capable of approximating any transfer function.

There is a variety of ANNs models found in the literature. As a universal approximation and having the simplest structure, the Multi-Layer Perceptron (MLP) is the most widely used ANN model. It has one input layer, where the influence factors (weather, calendar, etc.) are introduced and one output layer for the estimated value, the load data. Between the input and the output layers situate the hidden layers with neurons represented by nonlinear activation functions. Figure 2 shows a single layer perceptron's structure, which is the simplest of the feedforward layered structures.

![Figure 2 - Single perceptron structure](image)

Single Layer Perceptron (SLP) consists a layer of input neurons \( p_j, j = 1, \ldots, R \), associated weights \( \omega_{1,j}, j = 1, \ldots, R \), a weight \( \omega_{1,0} \) related to a constant \( p_0 \) termed bias, which equals to 1, an activation function \( g \), and an output \( c_1 \). The single perceptron compares the weighted sum of the inputs with the weighted bias, and the comparison result \( n \) goes through one activation function \( g \) to get the output. Lots of activation functions exist, such as step functions, linear functions, and sigmoid functions. For the approximation of a nonlinear model, the sigmoid function \( g(n) = \frac{1}{1 + e^{-n}} \) is the most commonly used for the hidden neurons. The weights \( \Omega_1, \{\omega_{1,j}, j = 0, \ldots, R\} \) of the neuron represent the efficiency of the synaptic connection. A negative weight inhibits its correspondent input, while a positive weight accentuates its correspondent input. The bigger the absolute value of the synaptic weight is, the more important the input is to the result. The extension to the SLP is the MLP, e.g. MLP with a hidden layer as presented in Figure 3.
The objective of the learning phase is to match the output of the MLP to the desired output, given the same inputs. This objective is reached by tuning the parameters $\Omega$ of the neural network, i.e., interconnection weights and bias. The learning procedures are grouped into three different categories: supervised learning, unsupervised learning, and hybrid learning. The essential element that differs supervised learning from other categories of learning is the presence of a “teacher”. In the supervised procedure (Figure 4), the teacher has some prior knowledge about the environment. Thus, when an input $P_i$ in the learning set is presented, the teacher is capable of providing the perfect output $y_i$. The “supervised neural network” presents the ANN trying to learn from the teacher by providing a close answer to the same input. The knowledge of the ANN is gained by the adjustment of his parameters under the influence of the deviation from the teacher’s answer, $\varepsilon_i$. 

**Figure 3** - One-hidden-layer network structure, where $P$ is the vector of variables $\{p_j, j = 0, \ldots, R\}$ and $\Omega$ is the vector of parameters $\{\omega_{i,j}, \omega_i, i = 1, \ldots, M, j = 0, \ldots, R\}$.

**Figure 4** - Supervised learning procedure.
The learning process stops when the difference between two responses is under a certain threshold, or the maximum number of iteration reached.

When to our load forecasting issue, let a set of \( N \) past historical data \( \{ P_i, i = t-1, \ldots, t-N \} \), such as time and weather, be the training set inputs, historical consumption samples \( \{ y_i, i = t-1, \ldots, t-N \} \) that are collected with a regular time step be the desired outputs, and \( \Omega \) be the set of ANN’s parameters. Minimizing errors in the training procedure, which is presented in Figure 4, solves prediction Figure 4:

\[
\min_{\Omega} \left\{ \frac{1}{2} \sum_{i=t-N}^{t-1} (y_i - f(P_i, \Omega))^2 \right\} \quad \text{(Eq.1)}
\]

### 3.3 Multi-Layer Perceptions and ANN models design

#### 3.3.1 Multi-Layer Perceptions and training process

The purpose of our work is to perform a 24 hour-ahead load prediction in a black-box fashion, i.e., without in operating prior knowledge into the model. Therefore, one must postulate a family of parameterized functions that have sufficient flexibility for implementing the unknown mapping between variables and the load to be predicted. Multi-Layer Perceptron (MLP) with one hidden layer in the neural network family is nonlinear functions that were proved to be universal approximators, i.e., to be able to approximate a sufficiently regular function to an arbitrary degree of accuracy [K. Hornik et al., 1989]. In addition, it was proved to be parsimonious. Therefore, one-hidden layer MLP is chosen for solving our problem.

A one-hidden layer MLP with \( M \) hidden neurons is a linear combination of \( M \) parameterized nonlinear functions called neurons. A neuron is a nonlinear, bounded function of a linear combination of its variables, usually an s-shaped (sigmoid) function such as the hyperbolic tangent. Therefore, the \( i^{th} \) “hidden neuron” of an MLP can be conveniently written as:

\[
c_i(P, \Omega_i) = \tanh \left( \sum_{j=0}^{R} \omega_{ij} p_j \right) = \tanh \left( \Omega_i^T P \right)
\]

where \( P \) denotes the \((R + 1)\) vector of the variables \( \{ p_j, j = 0, \ldots, R \} \) of the model, and \( \Omega_i \) denotes the vector of the parameters (or weights) \( \{ \omega_{ij}, j = 0, \ldots, R \} \) of hidden neuron \( i \) with \( i = 1, \ldots, M \). Note that variable \( p_0 \) is a constant equal to 1, termed “bias” of the hidden neurons. Thus, the MLP function can be written as:

\[
f(P, \Omega) = \sum_{i=1}^{M} \omega_i c_i(P, \Omega_i) = \omega^T C
\]

where \( \Omega \) denotes the set of the \((R + 1)M + (M + 1)\) parameters (figure 4) of the model, \( \omega \) denotes the \((M + 1)\) vector of weights of the linear combination, and \( C \) denotes the vector of the outputs of hidden neurons \( \{ c_i(P, \Omega_i), i = 1, \ldots, M \} \) with an additional component \( c_0 = 1 \), named the output bias.

Note that \( f(P, \Omega) \) is a linear function of the parameters of vector \( \omega \) and a nonlinear function \( c(\cdot) \) of the parameters of vectors \( \{ \Omega_i, i = 1, \ldots, M \} \). The nonlinearity with respect to some of the parameters is the origin of the parsimony of the models, the price to be paid being an increased complexity of training algorithms.

In our work in FLEXMETER project, \( f(P, \Omega) \) is the 24-hour ahead load prediction, \( P \) is the set of endogenous and exogenous variables, selected as described in subsection 4.3.a, and \( \Omega \) is the set of parameters of the model, which are estimated by training the neural network from a set of past measurements, the training set.
No attempt is made at designing a model that provides simultaneously predictions of all 24 hours of the next day, i.e., a neural model with 24 outputs [N. Amjady, 2009]. It is a very difficult task that cannot conceivably be performed by a parsimonious model, leading to an over parameterized model with dubious generalization capability.

The training set consists of \( N \) pairs \( \{\mathbf{P}_i, y_i\} \), where \( \mathbf{P}_i \) denotes the set of variables related to example \( i \), and \( y_i \) is the corresponding load measurement. The training process consists of estimating the parameters of the model \( \Omega \) by minimizing the distance between the output of the neural networks \( f(\mathbf{P}_i, \Omega) \) and load values \( y_i \) (Figure 5). For nonlinear regression problems such as investigated here, the usual distance is the least squares cost function:

\[
J(\Omega) = \sum_{i=1}^{N} \left( y_i - f(\mathbf{P}_i, \Omega) \right)^2
\]

Since the MLP function is nonlinear with respect to its parameters, the cost function is not quadratic with respect to the parameters of the model, so that the usual least squares fitting algorithm cannot be applied: one has to resort to nonlinear, iterative optimization algorithms, based on the computation of the gradient of the cost function. At each iteration, the gradient of the cost function is computed, and a parameter-update procedure that makes use of the value of the gradient of the cost function is applied, until convergence to a minimum of the cost function. This procedure can be accomplished efficiently by the so-called "backpropagation" algorithm.

In the present study, the gradient of the cost function was computed by backpropagation, and the parameter update procedure was the Levenberg-Marquardt algorithm, which is a second-order nonlinear optimization algorithm, the gradient of the cost function being computed by backpropagation\(^1\). These algorithms are implemented in the Matlab neural network toolbox that will be used in our work in order to validate proposed approach. A presentation of Matlab neural network toolbox can be found in https://fr.mathworks.com/products/neural-network.

\(^1\) The term backpropagation is frequently used for the computation of the gradient of the cost function followed by simple gradient descent, a training method that is not very efficient. This practice is avoided here.
3.3.2 ANN models design

i. Introduction

One of the purposes of machine learning in general and ANN in particular is the design of predictive models from data whenever prior knowledge on the process to be modelled is unavailable, or not accurate enough. For ANN models designs, we expect neural networks to provide nonlinear functions that can map past consumption values and exogenous variables at a given time to a future value of the consumption. The mapping is expected not only to explain the data from which the model is designed (termed training set), but also to generalize to test data set, i.e. to data that have not been used for designing the model. Therefore, in order to build an efficient model by machine learning, one must address two problems, as follows:

- Find the endogenous and exogenous variables that are relevant for predicting the power consumption. This is known as the variable selection problem.

- Find the appropriate model complexity given the available data. This is known as the model selection problem.

Note that the variable selection problem is not specific to nonlinear models, and must be addressed similarly for traditional linear models. By contrast, the model selection problem is ubiquitous for nonlinear models. The following universal property is observed experimentally and explained theoretically. A model that is not complex enough could be unable to explain the available data but able to generalize to previously unseen data. On the contrary, a model that is too complex fits the available data very well, but unable to generalize to previously unseen data, because such model is too specialized on the training data and fails to
capture the deterministic behavior of the process. This is known as the bias-variance dilemma. Complexity selection aims at solving the dilemma, i.e., finding the optimal model complexity from given the available data, in order to build an optimal predictive model.

A model design procedure aims at providing the model that has the best generalization capability, given the available data. This requires a search for the optimal model complexity. Since the number of parameters is linear with respect to the number of model variables and the number of hidden neurons, the design procedure has required discarding all candidate variables that are not relevant for the prediction task at hand, and finding the appropriate number of hidden neurons.

In the following, we describe the statistical tools used for variable selection and model selection.

**ii. Variable selection**

The principle of the method used for variable selection is the following [H. Stoppiglia et al., 2003]. To discriminate between irrelevant and relevant variables, a set of realizations of a random, hence irrelevant variable, called “probe variable”, are generated by randomly shuffling the components of the vectors of candidate variables. The probe variables, together with the candidate variables, are ranked in order of decreasing relevance by orthogonal forward regression [S. Chen et al., 1989]. The cumulative probability distribution of the rank of the probe variable is estimated, and the threshold rank \( r_0 \) is determined such that the probability for an irrelevant variable to be selected is smaller than a threshold \( \delta \) chosen by the model designer:

\[
p(r_{\text{probe}} < r_0) < \delta
\]

where \( r_{\text{probe}} \) is the rank of a realization of the probe variable, \( \delta \) is the risk of selecting a variable although it is irrelevant. Since the ratio of the complexity to the number of examples is crucial figure, a low value of \( \delta \) must be chosen if data are sparse in order to maintain a low complexity, while one may afford to choose a higher value of \( \delta \) if data are abundant. This variable selection method is named “filters”.

We denote by \( \xi_i \) the vector whose components are the \( N \) measured values of the i-th candidate variable \((i = 1, \ldots, n)\), and by \( y \) the vector whose components are the \( N \) measured values of the quantity to be predicted. If the variables have zero mean, the square of the correlation coefficient between the i-th candidate variable and the quantity of interest is given by [G. Dreyfus et al., 2003]:

\[
\cos^2 \varphi_i = \frac{\left(\xi_i^T y\right)^2}{\left(\xi_i^T \xi_i\right) \left(y^T y\right)}
\]

where \( \varphi_i \) is the angle between vectors \( \xi_i \) and \( y \) in the observation space. Ranking is performed by orthogonal forward regression, based on Gram-Schmidt or orthogonalization.

**iii. Model selection**

Since the cost function used for training neural networks is not quadratic with respect to the parameters of the model, it has local minima. The optimization algorithms being iterative, initial values of the parameters are required: they are generally drawn randomly from a probability distribution with zero mean and variance \( 1/R \) [G. Dreyfus et al., 2003]. The minimum reached by the optimization algorithm depends on the initial value of the parameters. Therefore, for a given number of hidden neurons, a variety of models can be obtained, each of these corresponding to a minimum of the cost function. Due to the bias-variance dilemma mentioned in above, it does not guarantees that the model corresponding to the global minimum of the cost
function should generalize better than a model pertaining to a local minimum. Therefore, a specific methodology must be used, as illustrated graphically in Figure 6.

![Diagram](image)

**Figure 6. Neural network selection procedure**

\{a,b\}: \{min, max\} number of hidden neurons; c: max number of initializations

The number of hidden neurons is increased from zero (linear model) to a maximum value (typically smaller than 10 in real applications); for each number of hidden neurons, many models are trained with different initial parameter values. The model with the best generalization ability will be selected. The assessment of the generation ability of the model is based on the Virtual Leave-One-Out (VLOO) method [G. Monari et al., 2000], [G. Monari et al., 2002], [V.N. Vapnik, 2000], as described below.

VLOO consists of withdrawing an example from the available dataset, training a model from the N−1 remaining examples, and computing the prediction error on the left-out example. The procedure is iterated by withdrawing each example in turn from the training set. The leave-one-out score is defined as:

\[
E_{LOO} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( y_i - f^{noti}(P_i, \Omega) \right)^2} \quad \text{(Eq.7)}
\]

\(f^{noti}(P_i, \Omega)\) is the model obtained when example \(i\) is withdrawn from the training set. If the number of examples \(N\) is large, this is a computationally expensive procedure, since the number of models is equal to the number of examples. However, if the model is linear with respect to its parameters, the exact value of the leave-one-out score is obtained by training a model from all training examples (withdrawing none) and using the Predicted REsidual Sum of Squares (PRESS) statistics [N.R. Draper et al., 1998] defined as:

\[
E_p = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( y_i - f(P_i, \Omega) \right)^2 \left( 1 - h_{ii} \right)} \quad \text{(Eq.8)}
\]

where \(f(P_i, \Omega)\) is the model trained from all examples and \(h_{ii}\) is the \(i\)-th diagonal element of the “hat matrix” \(H = Q(Q^TQ)^{-1}Q^T\), where \(Q\) is the observation matrix, i.e., the \((N, p)\) matrix whose general element \(Q_{ij}\) is the measured value of variable \(j\) in example \(i\) and \(p\) is the number of set of parameters. \(h_{ii}\) is termed the leverage.
of example $i$. If matrix $Q$ has full rank, $H$ is the orthogonal projection matrix onto the subspace defined by the columns of matrix $Q$. As a result, the leverages have the following properties: $0 \leq h_{ii} \leq 1$ and $\sum_{i=1}^{N} h_{ii} = p$.

Therefore, the leverage of example $i$ can be viewed as the proportion of the parameters of the model that is devoted to fitting the model to example $i$.

The model selection procedure is the following (as Figure 6): starting from a linear model (zero hidden neuron), the number of hidden neurons is gradually increased. For each number of hidden neurons, different models are trained with different initial values of the parameters; the VLOO score of each model is computed, and the model with the smallest VLOO score is selected. When, upon addition of a number of hidden neurons, the VLOO score starts increasing significantly, the procedure is stopped and the number of hidden neurons giving the minimum value of VLOO is retained. It may happen, however, that the VLOO does not vary significantly around the minimum value, in a certain range of number of hidden neurons. If such is the case, an additional selection criterion is taken into account. As mentioned above, the leverage of example $i$ represents the proportion of the parameters of the model that is devoted to fitting the model to example $i$. Therefore, a model that has one or more examples with leverages close to 1 is certainly very dependent on the specific measurement errors on these examples; thus, it will generalize poorly. On the contrary, a model whose leverages are close to their mean value $p/N$ will be influenced equally by all the examples, hence will have a good generalization capacity. Therefore, as a final selection criterion, the quantity $\mu$ is computed:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \frac{N}{p} h_{ii}$$  \hspace{1cm} (Eq. 9)

$\mu$ is always smaller than 1, and it is equal to 1 if and only if all leverages are equal to $p/N$. Among the models with similar VLOO scores, if any, the model with the highest value of $\mu$ is preferred.

### 3.4 Application and Numerical Results

In FLEXMETER project, real load data are collected in two demonstration sites, Malmö (Sweden) and Turin (Italy). During the first period of project, we have in possession more than one year data (05/2014-09/2015) collected from Malmö distribution networks by E.ON. These data were sampled every 5 minutes and saved in .csv files.

We illustrate the design procedure by applications to power consumption prediction of a building composed by about 20 living apartments. The objective is to forecast the load 24h ahead. In order to evaluate the precision of the model, we will forecast the load of a day in the past and compare with the exact load in that day. In theory, if we have load data of present date, it is possible to forecast the tomorrow load. However, this forecasting cannot be evaluated its accuracy because tomorrow real consumption is unknown.

A data set of 9 weeks (from 29/02 to 01/06/2015) is used in order as historical load to forecast the load on 02/06/2015. Weather data on this period are collected from website: www.wunderground.com. These weather data are publicly available and free. Three types of primary candidate variables are proposed: load variables, temperature variables and calendar variables. The first two types are numerical variables, while the calendar variables are categorical, but are transformed to numerical variables.

Load curve and exogenous variables that can influence to the load are presented in the Figure 7.
Figure 7. Load curve and exogenous variable
Training process and load forecast results are shown on the Figure 8 and Figure 9, respectively. The Mean Absolute Percentage Error (MAPE) of the model, which represents the average portion of the absolute forecasting errors to the real consumption values, is often used to evaluate the model. The MAPE is defined by:

$$MAPE(\%) = \frac{1}{N} \sum_{t=1}^{N} \left| \frac{y_t - \hat{y}_t}{y_t} \right| \times 100\%$$  \hspace{1cm} (Eq.10)

The MAPE of the obtained results is about 10%.

### 3.5 Conclusion

In this report, we have presented a new approach for load forecasting on distribution system using artificial neural networks. One hidden layer MLP structure has been used in our work. The methodology for ANN
models design addresses two problems: *variable selection* basing on the “filters” method and *model selection* basing on the VLOO score. The proposed methodology has been applied to forecast load of a building in Malmo site with a desired accuracy.

Currently, algorithm is implemented in MATLAB environment with its neural network toolbox. For a whole application in Energy Aggregator platform, it requires a development in a programming language such as C++, Java ou Python with an available neural network library. In the future, we can apply the proposed method to forecast the load in Turin site (when data from IREN are available). Associated with algorithm of NIALM developed in T4.2, we can envisage forecasting load consumption for apartment in a building. Bill prediction for prosummers can be realized by exploiting the results of load forecasting.
4 Distributed Generation forecasting

Due to the intermittency of Distributed Energy Resources, under and over-generation become a grid management problem. Electrical Distributed Generation Forecasting will become fundamental for grid management to avoid displacement between demand and supply of energy.

PV system power production from solar irradiance is well known in literature as a function of the received incident solar irradiance. So, our effort has been concentrated on the prediction of the global solar radiation incident on a horizontal surface. Broadly speaking two kinds of approaches are possible: Model-Based and Model-Free.

Model-Based approaches try to build the physical model under consideration because the analytical terms of clear sky are well known. However, a combination between theoretical clear sky-model with weather data is still needed: the weather variables that influence the resulting irradiance are not taken into account by this model.

By contrast Model-Free approaches bypass the physical clear-sky model altogether. They cast irradiance prediction into a supervised regression problem. This approach exploits a full range of predictors / features such as past and present weather data, day ahead forecasts, time of the day, day of the year, etc. to make prediction of the target solar irradiance.

This chapter is focused on solving the problem of irradiance prediction as an instance of sequence-to-sequence learning problem. The predicted solar irradiance can be used as input to a different PV system models to estimate the day ahead energy balance of a geographical Substation Unit area.

This system will be one of the component of the Energy Aggregator Platform presented in D4.3 and will be an input for different energy management services, like the Demand Side Management. First, an introduction to general forecasting systems will be presented followed by a specific description of the proposed Generation Forecasting model. TIM has filed a patent as a direct outcome of the present work.

4.1 Analysis of predictor variables - the weather data

The present approach falls under the Model Free category. Our basic requirement was to exploit weather information sources publicly and freely available, such as Forecast.io (now renamed DarkSky), OpenWeatherMap, and Weather Underground services. We choose Forecast.io because his APIs are consistent for both historical weather observations and weather forecasts. So we can find the same variables provided as measured quantities in historical data, or as day ahead predicted measurement. Forecast.io API follows this format:

https://api.forecast.io/forecast/APIKEY/LATITUDE,LONGITUDE,TIME

The API variables are:

- APIKEY is a developer’s personal API key
- LATITUDE geographic latitude of a location in decimal degrees
- LONGITUDE geographic longitude of a location in decimal degrees
- TIME (optional) date of a past day for which we want weather data. If it isn’t specifiedation, the request return a day-ahead forecast of the weather variables
We selected a subset of the hourly weather variables\(^2\) returned by the Forecast.io service that are the most relevant predictors for solar irradiance:

- **time**: UNIX time
- **summary**: summary as a human readable text
  
  Ex. “Overcast”, “Clear and windy”, “Partially Clouded”...
- **temperature**: temperature as a numerical value
- **humidity**: relative humidity as numerical value between 0 and 1 (inclusive)
- **cloud-cover**: percentage of sky occluded by clouds as numerical value between 0 and 1 (inclusive)

### 4.2 How to exploit past and future data

Features in Machine Learning terminology are the independent variables. The target variable(s) are obtained by regressing over the features. In our case, the solar irradiance is the target variable while time and weather quantities are the features.

It may be possible to solve the problem with a supervised learning problem. We simply choose our preferred ML algorithm and train the system. There are plenty of available algorithms, such as Ridge Regression, Support Vector Regression, Elastic Nets, Neural Networks, Gaussian Process Estimation, etc.

The supervised regression model will be trained with past observations and during the inference it will provide the irradiance prediction based on the corresponding weather forecast. However, even though the weather variables bring the most part of the information necessary for the prediction, it is possible to increase the accuracy of the prediction by conditioning on past weather data forecasts, like auto regression on sequential data.

Sequential data contain some intrinsic imprinting on its generating statistical process. ML methods try to discover such structure, and estimates statistics about it to make accurate predictions. Many statistical methods are employed in these fields like: ARIMA, GARCH, Triple Exponential Smoothing, etc.

Again, we would train our model with past irradiance observations, but this time during inference we would provide the trained model with the most recent observations, weather data AND the corresponding solar irradiance. The model should be able to predict the near future based on the near past only.

We can see that the two approaches described are alternative to one another. Either we make solar irradiance dependent on weather data, and during inference we provide such weather forecast only; or we make solar irradiance AND weather data auto-regress on themselves, and during inference we provide the most recent data to predict the very near future.

Intuitively we may think that the first approach should give better results. If the model really uncovers the deterministic function that maps weather data on solar-irradiance than such model is enough for reliable irradiance predictions, save the usual noisy terms. That would be true if the predictor variables were really trustworthy. But in fact weather forecasts are themselves forecasted as the name implies, and as such are not ground truth and carry as degree of uncertainty which is outside our scope / control.

Given the above, it would be beneficial to join the strengths of the respective models. We would like to make solar irradiance depend on weather forecast, yet we acknowledge that such weather forecast is not

---

\(^2\) Forecast.io provides hourly day-ahead forecasts for 48 hours; minutely forecasts for the next hour; daily forecasts for the next week. We only used hourly forecasts.
completely reliable, and so we condition the irradiance prediction on past irradiance observations as well. The model will condition its inference on the recent past observations, actual measured solar irradiance and weather data, AND the day-ahead weather forecast. It will be up to the model to balance these complementary information and provide the most reliable irradiance prediction. During training the model will automatically learn three tied objectives:

- exploit day-ahead weather forecasts; like a regression problem
- exploit recent past weather and irradiance observations and historical weather forecast; like an auto-regression problem
- weigh the relative importance / reliability of the two predictors above.

Sequence-to-sequence learning

There are not many ML approaches that allow both regression and auto-regression. In fact the only one we know of is sequence-to-sequence (seq2seq) with recurrent neural networks (RNN). In very recent years seq2seq models have been really successful in many natural language processing tasks, such as language translation or even image captioning. A basic seq2seq model, as introduced in Cho et al., consists of two RNNs called encoder and decoder. The network topology is made such that the decoder is conditioned on the encoder but also on some other input.

In Figure 10, the encoder compresses its input sequence $x_1, \ldots, x_T$ into a context vector $C$ represented by its final hidden state. Such context vector is used by the decoder while processing its own sequence $y_1, \ldots, y_T$. In addition, encoder and decoder may be fed with sequences of completely different lengths.

In seq2seq the model is trained end-to-end. This means that encoder and decoder are trained jointly on the prediction task; there is not separate pre-training and later a combination of the two. End-to-end means that the training signal is propagated directly from the loss function to all the network’s parameters. In the basic seq2seq model the encoder RNN processes its input sequence, and the final hidden state of the encoder
represents the “compressed” information of the whole sequence.

This is a natural feature of RNNs where their hidden state depends on the current input and the previous hidden state. In other words RNN are auto-regressive machines. Since the final hidden state of the encoder is a compressed representation of the whole input sequence, such state is used to initialize the hidden state of the decoder RNN. This way the decoder RNN is conditioned on its own input AND the encoder’s input by proxy of the hidden state.

This basic seq2seq model has later been augmented with a so called attention mechanism introduced by Bahdanau et al. In their language translation task they conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing the seq2seq model to automatically attend / focus on parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. The distinguishing feature of the attention mechanism from the basic encoder–decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector represented by the encoder’s hidden state. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. The key intuition is that the attention mechanism can be models as a complementary network that learns to soft-weight the elements of the input sequence and can be jointly trained end-to-end along with the encoder and decoder networks. In the typical setting the attention network is a feedforward neural network whose output is a soft-max over the sequence vectors of the encoder. The soft-max output of the attention network is the used with element-wise product on the encoder’s vectors to weight their respective relevance given the context of the decoder. In fact the attention network is itself conditioned on the current hidden state of the decoder, in other words the attention depends on what the decoder is doing right now.

![Figure 11 – By-directional RNN](image)

In Figure 11, the encoder is made by a bi-directional RNN depicted as two reverse RNNs. The output sequence $h_1,\ldots,h_T$ is weighted by the attention network (not depicted) through the weights $\alpha_{t,i}$. The presence of the indicator ‘t’ in each weight $\alpha_{t,i}$ remarks the dependence for the weights for each time step of the decoder, here depicted at the top.

We recall that a soft-max is a typical non-linearity applied on the last layer of a classification network where the values represent the probabilities of each class under the observations. In this setting these are the probabilities of the various encoded vectors. Being probabilities they sum to one, but the nice thing is that they represent a soft multimodal distribution: not just a single vector is one and all other zero (that would be hard attention), but maybe two or even three or more vectors are relevant for a particular output.
4.3 seq2seq for irradiance forecast

Our objectives were:

- exploiting day-ahead weather forecasts; like a regression problem
- exploiting recent past weather and irradiance observations; like an auto-regression problem
- combining and weighting the relative importance / reliability of the two predictors above.

In fact, with seq2seq, we

- process the recent past weather and irradiance observations through an encoder;
- pass the day-ahead weather forecast to a decoder;
- introduce an attention mechanism that weights the relative importance of the encoder’s vectors and condition the decoder on this augmented information.

4.4 General architecture

![Diagram of General Architecture of the Developed Method]

*Figure 12 - General Architecture of the Developed Method*
4.5 Dense embedding of discrete variables

The first thing to do when setting up any ML problem is to and preprocess the data. We already mentioned that Forecast.io provides very many weather quantities. The first thing to do is to filter out all the variables that are irrelevant for the target. For example sunrise-time, sunset-time, nearest-storm-distance, etc. are all variables that are not relevant for irradiance prediction. For the remaining variables, many are correlated and only some of them needs to be considered. For example humidity and dew-point are highly correlated; likewise temperature and feel-like-temperature. Finally some variables would have been relevant but had to be removed because their values were often missing in the historical observations. For example the atmospheric pressure is an important predictor but unfortunately the records provided by Forecast.io have many of this field missing, rendering this quantity so sparse to make it too unreliable for consideration.

So in the end we have considered these features: time, temperature, humidity, cloud-cover, and summary. It's immediately clear that these variables are a heterogeneous bunch. Some of them are continuous, such as temperature, humidity, and cloud-cover. Some are discrete, like summary, which consists in a text label of the weather condition. Finally ‘time’ is a multi-value onto itself. We may naturally be inclined to think time as a continuous variable, and indeed that's somehow correct. However in ML analysis data need to be preprocessed, which also means to be adapted to the particular task at hand to facilitate learning. In particular we notice that the most relevant time information for the prediction of solar irradiance is the point time of the day, and the day of the year. In our context the point time of the day reduces to one of the 24 daily hours, as ‘hour’ is unit at our disposal from Forecast.io. Indeed any finer interval would not make much sense due to the variance in weather forecasts. Therefore our time information is actually a discrete variable that denotes one out of 24 hours of the day, and another discrete variable that denotes one of the 365(6) days of the year. So in the end we have three continuous variables and three discrete variables.

Pioneered by Mikolov et al., in 2013 and applied to natural language processing, the word2vec technique serves as a bridge between a discrete, sparse representation and a continuous, dense one. Word2vec learns a mapping between the words of a dictionary, which are countable and discrete, into a continuous Euclidean space, called embedding. The beauty of the word2vec is twofold:
it automatically learns the mapping from the sparse discrete variable to the embedded Euclidean space, where distances\(^3\) become meaningful. For example, in this embedding we discover semantic syllogisms as simple algebraic operations:

Ex. king - queen \(\approx\) man - woman

secondly, the continuous embedding allows training with efficient flow propagation of stochastic gradient descent (SGD) and its more sophisticated siblings, Adagrad, RMSprop, Nesterov, Adam, etc., which are all variation on the basic SGD.

Coming to the task at hand we designed the model to firstly learn an embedding for each of the discrete variables: hour-of-day, day-of-year, and weather-summary. The resulting embeddings are dense, continuous representations that are jointly concatenated with the other continuous variables, temperature, humidity, cloud-cover. We started with 3 continuous and 3 discrete variables and after the embedding layer of our network we’re left with 6 coherent continuous variables. The purpose of using the embedded is to map the sparse data into a dense space in order to optimize training phase and efficiently manipulate data.

4.6 What to encode and what to decode

The leitmotif in this document is the application of seq2seq for solar irradiance prediction. We’ve already mentioned that the decoder is fed with the past observations, both the observed weather data and measured irradiance. We have modeled the encoder as a bi-directional recurrent network. In particular, we have chosen Gate Recurrent Units (GRU) over Long Short Term Memory (LSTM) networks because GRUs provide the same performance of LSTMs but with much fewer parameters and less computational cost\(^4\).

A sequence of 72 observations of the past three days is given to the bi-directional GRU encoder. In the whole, this data is a 3-dimensional tensor whose shape is \([\text{mini-batch-size, sequence length, feature-size}]\). The first dimension is the size of the mini-batch of data, that is the samples over which to make a single step of SGD training. The second dimension is the sequence of data for the RNN. We have chosen a length of 72 that corresponds to the past three days. When deciding on sequence length we need to trade-off the computational cost of longer sequences vis-à-vis their real increased information. Intuitively, longer sequences are better but the information gain of events much further in the past become negligible. Finally the last dimension is the feature-size, i.e. the independent variables. Overall we have 6 weather and time variables, plus the irradiance itself, which as past info is not a target unknown but a measured quantity. So the dimension of the features is \(6 + 1 = 7\).

As said this 3-dim tensor is fed to the bi-directional GRU encoder. Bi-directional RNN are nothing else than two sibling RNNs, one processes the sequence normally and the other processes the same sequence but in reverse order. The first normal RNN uncovers the effects due to some causes occurred in the past. The reverse RNN on the other hand, uncovers the causes of some effects observed in the future\(^5\). The two RNNs are independent of each other, and their respective outputs and final hidden states are typically either summed or concatenated. In this work they are summed to avoid duplicating the number of parameters to train.

\(^3\) Cosine distance is usually preferred over Euclidean distance

\(^4\) For the difference between GRU and LSTM over vanilla RNN the reader is encouraged to refer to the relevant articles and books that abound on the web.

\(^5\) Reversing the sequence is like observing the future and going backward.
So in the end the result of the encoder layer is a single hidden state that coalesces the two bi-directional RNNs, and a sequence of output vectors, as many as the input sequence length, 72 in our case. It is this sequence that will provide the context over which the attention network will focus on. The attention network is a feed-forward network whose input is the encoder output vectors and the decoder current hidden state, and whose output is a weighted summary of the encoder vectors through a soft-max non-linearity. For details refer to Bahdanau et al.

Finally, let’s shed some light on the decoder architecture. The decoder is a single GRU RNN (not bi-directional). The hidden state of the decoder is initialized with the hidden state of the encoder. The input to the decoder is the sequence 24-hour day-ahead weather forecast of the day we want to predict the irradiance. Such sequence is a tensor shape \([\text{mini-batch-size}, \text{sequence length}, \text{feature-size}]\). Again, the first dimension is the size of the mini-batch of data. The second dimension is the sequence of data for the decoder RNN which in this case is the 24 hour ahead weather forecast. The third dimension is the feature-size, i.e. the independent variables. Contrary to the encoder which is fed with both weather variables and the irradiance, the decoder only receive the weather info, while the irradiance is its target. In fact the network is trained with a L2 loss\(^6\) on the final output layer to reconstruct the true irradiance.

Note that the decoder RNN receives two inputs: the 24 hour weather forecast and the output of the attention network. As said above the attention network is itself conditioned on the current hidden state of the decoder, therefore the two networks are mutually dependent letting the model learn a very complex distribution.

### 4.7 Some implementation details and results

The seq2seq model has been implemented in Python with the TensorFlow library. Weather data has been retrieved through Forecast.io web API and saved in local CSV files for later processing. Solar irradiance data has been provided by the Politecnico di Torino as measured by local light sensors. The combined data has been processed with Pandas, another well-known Python library. Data has been resampled to uniform 1-hour bins, cleansed of missing values, and standardized to zero-mean, unit-variance.

The final training set consisted of some more than 25K observations over 4 years, from 2012 through 2015. After cleansing the dataset of missing values, we were left with 27400 observations. The data set was split in a first 90% for training and the last 10% for testing.

Batch size during training was set to 32, sequence lengths were 72 for the encoder and 24 for decoder. Hidden size for all RNNs was set to 30, dropout was applied after each RNN with a keep probability of 0.7 during training, and no dropout during testing. The embeddings were all 3 dimensional; L2 loss was applied on the final layer; Adam optimizer was used with standard weight decay. The network was trained over 10 epochs. The final mean-square-error at training was 0.66 as the sum for the 24 hours. MSE at training was 1.08. Results are presented in Figure 14.

---

\(^6\) L2 loss a.k.a. mean-square-error is the typical loss for homoscedastic normally distributed variables.
Figure 14 - Training and Testing Results
5 Conclusions

This document describes the algorithms of short-term load and irradiance forecasting in distribution networks. These forecasting tools are based on the application of artificial neural networks. For the load forecasting, we propose a methodology for MLP models design that are validated by using real data collected in Malmo (Sweden). For generation forecasting, the named sequence-to-sequence learning technique with recurrent neural network is chosen to forecast day ahead irradiance pattern. The algorithm has been implemented in Python and given accurate forecasting results.

Load and generation forecasting are components of the Energy Aggregator (EA) Platform. Their outputs serve other components of the platform, such as demand-response, with load consumption and energy production in the day ahead. The output could be also exploited directly by external services of DSOs. Moreover, information from load and generation forecasting are useful for purchase planning of distribution system managements.
Bibliography


G. Monari and G. Dreyfus. Local over fitting control via leverages. Neural Computation, 14:1481-1506, 2002
