Estimating Harvestable Solar Energy from Atmospheric Pressure Using Deep Learning

Tereza Paterova*, Michal Prauzek
Faculty of Electrical Engineering and Computer Science, VSB - Technical University of Ostrava, Ostrava, Czech Republic
tereza.paterova@vsb.cz

***Abstract*—This article focuses on applying a deep learning approach to predict daily total solar energy for the next day by a neural network. Predicting future solar irradiance is an important topic in the renewable energy generation field to improve the performance and stability of the system. The forecast is used as a support parameter to control the operation duty-cycle, data collection or communication activities at energy-independent energy harvesting embedded devices. The prediction is based on previous hourly-measured atmospheric pressure values. For prediction, a back-propagation algorithm in combination with deep learning methods is used for multilayer network training. The ability of the proposed system to estimate the daily solar energy is compared to the support vector regression model and to the evolutionary-fuzzy prediction scheme presented in previous research studies. It is concluded that the presented neural network approach gave satisfying predictions in early spring, autumn, and winter. In a particular setting, the proposed solution provides better results than a model using the support vector regression method (e.g., the MAPE value of the proposed algorithm is 0.032 less than the MAPE value of support vector regression method). The time and computational complexity for neural network training is considerable, and therefore it was assumed to train the network on an external computer or a cloud, where only the network parameters have been obtained and transferred to the embedded devices.**

***Index Terms*—Energy management; Environmental monitoring; Neural networks; Prediction algorithms.**

I. INTRODUCTION

Energy-independent embedded systems are specially designed for deployment sites [1], where it is not possible to connect such devices to a power grid. A common solution for supplying energy to these devices is using batteries to satisfy energy demands. Due to the frequent deployment in inaccessible locations, this solution has several drawbacks. Another solution includes a combination of rechargeable batteries and an energy harvesting subsystem. This approach, compared to a solution without energy harvesting, needs an energy management strategy, where a knowledge of the future energy availability can help to design a more efficient energy management strategy. This approach provides important input parameters represented by a future energy estimation and it allows the better scheduling of data collection tasks and the system can work efficiently in terms of the distribution of computational power [2], [3].

In general, predicting the future solar irradiance is an important topic in the renewable energy generation field. It is used to improve the planning and operation of the photovoltaic systems [4]. The accuracy of the prediction improves the performance and stability of the systems [5]. The solar availability can be predicted using many methods, such as statistical methods, artificial neural networks (NN), support vector machines, and autoregressive moving averages [4]. Besides these methods, other methods can also be used (regression tree, random forest, gradient boosting, and many others) in the context of the estimation of the future solar energy [6]. In general, these principles can be transferred to the energy harvesting embedded system area to estimate the future obtained energy when the estimation is based on an internal atmospheric pressure sensor.

Machine learning (ML) methods, such as deep NN, are suitable for estimating future solar energy because they add complexity to a model without specifying what form the variation should take and allow the extraction of high-level features [7]. To improve the prediction performance, some authors proposed using hybrid models or an ensemble forecast approach [6], [8]. Deep learning enables the computational model (e.g., a multilayer NN) to create and remember a new representation of the input data [9]. Nowadays, models trained through deep learning have succeeded in the fields of speech and image recognition, object detection, time regression rows, etc. [10].

Deep learning provides tools for solving complex problems using multilayer NN. A fusion of NN and deep learning approaches lead to Deep NN, which can be defined as NN with several (at least three or more) hidden layers [9]. The NN ability to solve more complex tasks increases with the increasing number of layers, but there are increasing demands on the complexity and time to train such a NN. Many problems solved by deep NN are based on mapping input data vectors to output vectors. The fulfilment of these tasks is conditioned by a sufficiently complex NN topology

Manuscript received 5 February, 2021; accepted 19 May, 2021.

This work was supported by the project No. SP2021/29, "Development of algorithms and systems for control, measurement and safety applications VII" of the Student Grant System, VSB-TU Ostrava. This work was also supported by the European Regional Development Fund in the Research Centre of Advanced Mechatronic Systems project, project No. CZ.02.1.01/0.0/0.0/16_019/0000867, within the Operational Programme Research, Development and Education. This work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant No. 856670.

with a large task-representing data set consisting of a set of input vectors and their corresponding output vectors. Tasks that cannot be implemented using input-to-output mapping are now largely beyond deep learning [10].

This paper focuses on the application of deep learning and NN approaches, especially on deep feedforward networks, to forecast the next-day solar energy availability from the atmospheric pressure in an energy harvesting embedded system. A back-propagation method is used to train a multilayer NN in combination with deep learning methods, such as optimalization, batch normalization, and initialization of weights. The algorithm implementation was executed in the C programming language. The results of each combination are compared to previous published approaches and an overall evaluation is provided.

II. DEEP FEEDFORWARD NETWORKS

The aim of the deep feedforward NN is to approximate a complex function f^* by defining the mapping of the input vector x to the initial vector y. The mapping has the form of (1), where Φ represents the set of parameters resulting from the functional approximation [10]

$$y = f\left(x, \Phi\right). \tag{1}$$

The deep feedforward NNs are formed by the hidden layers. The input and output layer are defined by the structure of the input and output data. The properties of a NN are determined by a topology, number, and width of the hidden layers. With the increasing number of layers and neurons, the complexity of the network training increases, but the ability of NN to solve more complex tasks also increases [11], [12].

One of the deep learning methods is the gradient descent algorithm, which is used for NN learning. The gradient descent is a method of minimizing the purpose function $J\left(\Phi\right)$ by parameterizing the model parameters $\Phi \in R^n$. The parameterization is realized by updating the parameters in the direction opposite to the gradient of the objective function $\nabla_\Phi J\left(\Phi\right)$ [13].

In general, there are three basics configurations of the gradient descent - a batch gradient descent (an ordinary gradient descent), a stochastic gradient descent, and a mini-batch gradient descent - which differ according to the volume of data from the total dataset that is used to calculate the gradient of the objective function [14]. The most used gradient algorithm for descent optimization is back-propagation (BP). Generally, BP is a type of batch gradient descent, but it can be converted to the other two types. The choice of type depends on the application, time requirements, availability, and data content [10].

The form of the stochastic gradient descent is shown in (2)

$$\Phi_{new} = \Phi - \alpha \times \nabla_\Phi J\left(\Phi; x_d; t_d\right). \tag{2}$$

The parameter is updated for each training sample x_d and its corresponding result t_d [15].

The stochastic gradient descent performs frequent updates with a high deviation, which causes high fluctuations in the purpose function. However, these fluctuations may allow them to escape from the local minimum, which may be advantageous, especially for the non-convex surfaces of the purpose function. On the other hand, it slows down convergence to the exact minimum, though this can be removed to some extent by optimizing the learning coefficient α (with a very good choice of α, the same convergent behavior can be achieved as with the batch gradient descent) [14]–[30]. The learning coefficient α determines the length of the step that is in progress when updating to reach the (local) minimum. In other words, we follow the direction of the steepest slope of the surface of the purpose function in steps [30]

$$\Phi_{new} = \Phi - \alpha \times \nabla_\Phi J\left(\Phi; x_{d;d+n}; t_{d;d+n}\right). \tag{3}$$

The Mini-batch gradient descent is a compromise between the two approaches. This gradient descent method performs an update for each mini-batch of training data; therefore, the entire training data set is not used, but only a certain part of it, see (3) [16], [17]. Unlike the stochastic gradient descent, this method is not as sensitive to changes in hyper-parameters, especially the learning rate used during optimization, which leads to a reduction in fluctuations and thus to a more stable convergence [14].

A. Back-Propagation

Back-propagation (BP) is a commonly used algorithm for training the multilayer feed forward NN [16]. There are various learning parameters, such as the learning rate, momentum or activation function which can improve the BP learning algorithm. These improvements significantly speed-up NN convergence and avoid convergences in the local minimum [18]. These approaches are described further in the subsections "Initialization of weights" and "Optimization methods".

The BP method requires a set of training data consisting of ordered pairs of vectors, where one pair is formed by a vector of input values $\hat{x}_j$ and the other pair is formed by a vector of required outputs $\hat{y}_j$ for input $\hat{x}_j$. The set contains N of these training vector pairs [10].

The algorithm of the BP method consists of the following steps. Firstly, forward transmission is performed for the input-output pair of vectors $\left(\vec{x}_d, \vec{t}_d\right)$ and the vector of the calculated output values of NN $\vec{y}$ (a_j and y_j for each neuron) is stored. Secondly, the reverse transmission is realized and the result $\frac{\partial E_d}{\partial \omega_{i,j}^l}$, where E is error function and ω is set of parameters resulting from the best functional approximation, is stored. This step can be divided into the following sub-steps:

1. Calculate δ_j^L, i.e., calculate the auxiliary variables δ for the output layers L, see (4)

$$\delta_j^L = \frac{\partial E}{\partial a_j^L} \times \left(t_{d,j} - y_j^L\right). \tag{4}$$

2. Perform a calculation of δ_j^k for all neurons across all remaining layers (5), where F is the intrinsic potential of the neuron

$$\delta_j^k = \frac{\partial F\left(a_j^k\right)}{\partial a_j^k} \sum_{l=1}^{r_{k+1}} \left(\omega_{j,l}^{k+1} \times \delta_j^{k+1}\right). \quad (5)$$

3. Calculate the individual error E_d according to $\omega_{j,l}^k$.

– For hidden layers and the output layer by (6)

$$\frac{\partial E}{\partial \omega_{i,j}^k} = \delta_j^k \times y_i^{k-1}. \quad (6)$$

– For the input layer by (7)

$$\frac{\partial E}{\partial \omega_{i,j}^k} = \delta_j^k \times x_{d,i}. \quad (7)$$

4. Perform steps *a*-*c* for all input-output vector pairs. Combine the individual errors $\frac{\partial E_d}{\partial \omega_{i,j}^k}$ to obtain the total error $\frac{\partial E\left(X,\Phi\right)}{\partial \omega_{i,j}^k}$ for the whole set of training data (input-output vector pairs) by (8)

$$\frac{\partial E\left(X,\Phi\right)}{\partial \omega_{i,j}^k} = \frac{1}{N} \sum_{d=1}^{N} \left(\frac{\partial E_d}{\partial \omega_{i,j}^k}\right). \quad (8)$$

5. Update the weights according to the learning coefficient α by (9)

$$\omega_{i,j_{new}}^k = \omega_{i,j}^k - \alpha \frac{\partial E\left(X,\Phi\right)}{\partial \omega_{i,j}^k}. \quad (9)$$

6. Repeat the entire procedure until the total NN error is less than the specified accuracy, or until the completion of a specified number of iterations [18], [19].

B. The Problem of the Vanishing-Exploring Gradient

The problem of the vanishing-exploring gradient especially occurs in recurrent and multi-layered NN. The problem concerns the updating of NN parameters during learning. The principle of the vanishing gradient lies in the decreasing value of the partial derivative of the error function of a particular parameter during the learning process, which leads to a complete cessation of convergence. This causes the parameters in the remote layers to change significantly more slowly than in the layers close to the output layer. The simplest solution is to choose an activation function whose derivative is greater (e.g., ReLU). On the other hand, if the learning coefficient is chosen incorrectly, using the ReLU activation function can cause divergence. This problem can be largely eliminated by applying optimization methods and the appropriate initialization of parameters or pre-training [20], [21].

C. Initialization of Weights

It is necessary to set the NN parameters before starting the learning algorithm with the proper selection of the initialization method. Those parameters should correspond to the chosen activation function (e.g., for sigmoid function it is suitable to initialize weights in the interval (0, 1), for hyperbolic tangent function, the interval (-1, 1)) [22] is more suitable.

There are many weight initialization configurations, such as the Rectified Linear Unit (ReLU) activation function, hyperbolic tangent function, and sigmoid function. For the ReLU activation function, the correct weight initialization is not a major problem. For any nonnegative number, the gradient is equal to the internal neuron potential. For a negative number, the gradient is equal to 0 or the parameter α. However, for faster convergence, it is advisable to initialize the weights with numbers from the interval defined by (10), where n is the number of inputs in layer k

$$\left(n^k \sqrt{\frac{2}{n^{k-1}}}, n^{k-1} \sqrt{\frac{2}{n^{k-1}}}\right). \quad (10)$$

For ReLU, Xavier initialization can be used with a normal distribution in the interval defined in (11) [23]

$$\left(0, \sqrt{\frac{2}{n^k + n^{k+1}}}\right). \quad (11)$$

For the hyperbolic tangent function, Glorot's initialization with normal distribution in the interval defined by (12) is used [24]

$$\left(-\sqrt{\frac{6}{n^k + n^{k+1}}}, \sqrt{\frac{6}{n^k + n^{k+1}}}\right). \quad (12)$$

For the sigmoid activation function, the uniformed Xavier initialization method in (13) is used [22]

$$\left(-\sqrt{\frac{2}{n^k + n^{k+1}}}, \sqrt{\frac{2}{n^k + n^{k+1}}}\right). \quad (13)$$

D. Optimization Methods

Optimization methods compensate for the problem of an increasing or decreasing gradient (i.e., a situation where either the gradient converges rapidly to zero or, conversely, diverges to infinity, such as the problem of a vanishing-exploring gradient) [25].

Adam is an algorithm for optimizing a stochastic object function based on the Jacobian calculation. The algorithm is computationally efficient with no extensive memory requirements and it is invariant to the diagonal scaling of gradients. This approach is based on the calculation of individual learning NN coefficients from the estimation of the first and second moment of the gradient [22]. The algorithm combines the advantages of the AdaGrad method, which is suitable for solving problems with small gradient values [5], and the RMSProp method, which is suitable for

the online learning mode (stochastic gradient descent) with non-stationary settings [26]. The advantage of Adam's algorithm is that the magnitudes of the parameter changes are invariant to the gradient scaling. It does not require a stationary object and it is also suitable for cases where the gradient takes on small values [22].

Nesterov Momentum is a modification of the traditional momentum method. The gradient is calculated at the point that would be reached after performing a previous step β times. This point is obtained by summing the original values of the parameter with a multiple of the previous vector $\overline{V}$ (initialized to zero) with the friction parameter β [25]. RMSProp is one of the adaptive learning rate methods, i.e., methods that optimize the learning coefficient during NN training. It uses "signal-to-noise" normalization with an absolute magnitude of the gradient in combination with exponential averaging [25], [26].

E. Batch Normalization

The training of deep NN is complicated by the fact that the inputs of all neurons in the hidden layers and the output layer are affected by the parameters of the previous layer's neurons. During training, the parameters of neurons, and thus the activation variables and inputs of neurons, are changing. Changes in neuron inputs in the previous layer are transmitted to the current and subsequent layer. This propagation causes instability leading to reduced convergence or even divergence in the deeper layers. Such a problem is described as a change in the distribution of layer inputs. The change in layer distribution is referred to as an internal covariance shift [27], [28].

The aim of the batch normalization is to reduce the internal covariance shift by adding a normalization layer. The normalization layer can theoretically be placed before the activation function application (this is pre-activation batch normalization) or at the output of the activation function (post-activation batch normalization). Practically, only the pre-activation version is usually used [27]. Batch normalization allows the application of a larger learning coefficient without the risk of divergence. It also performs model control and thus eliminates the need to use the Dropout control [29]. The normalization pre-activation layer is added for each neuron according to (14)–(18):

$$\mu_i = \frac{\sum_{r=1}^{m} v_i^{(r)}}{m}, \tag{14}$$

$$v_i^{(r)} = \sum_{j=1}^{k} \omega_{j,i} \times x_j^{(r)} + b_i, \tag{15}$$

$$\sigma_i^2 = \frac{\sum_{r=1}^{m} \left(v_i^{(r)} - \mu_i \right)^2}{m}, \tag{16}$$

$$\overline{v}_i^{(r)} = \frac{v_i^{(r)} - \mu_i}{\sqrt{\sigma_i + \varepsilon}}, \tag{17}$$

$$a_i^{(r)} = \gamma \times \overline{v}_i^{(r)} + \beta_i, \tag{18}$$

where $v_i^{(r)}$ is the r-th sample of the non-normalized potential of the i-th neuron, $\overline{v}_i^{(r)}$ is the r-th sample of the normalized potential of the i-th neuron, $\overline{a}_i^{(r)}$ is the r-th sample of the output of the normalization step of the i-th neuron, ε is the offset to prevent the failure of the power factor calculation selected in the order of 10^{-8}. β_i, γ_i are neuron parameters introduced by batch normalization. These layers are always applied to one batch of samples of size m [27].

III. DEEP LEARNING EXPERIMENT

The aim of the presented experiment was forecasting a solar energy availability for the following day by using a deep learning algorithm with a multilayer perceptron. A back-propagation algorithm was used to train multilayer NN in combination with deep learning methods, such as optimalization, batch normalization or the initialization of weights. Each of these deep learning methods has several configurations which will be compared. The deep learning experiment algorithm for one configuration of deep learning methods is shown on Fig. 1.

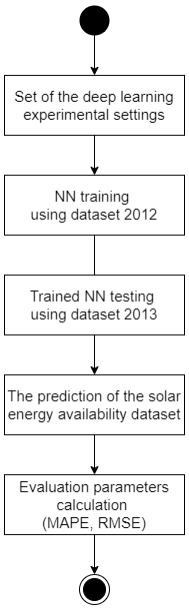

Fig. 1. The deep learning experimental algorithm.

First, a test configuration of deep learning methods is set. Then the NN training is performed using the dataset 2012. Subsequently, the trained network is tested using the dataset 2013, which results are the prediction of the available solar energy. Then the evaluation parameters of the resulting dataset are calculated. These parameters are used for comparison to other test configurations of deep learning methods, which are performed.

Outputs of the forecast algorithm were validated by a Mean absolute percentage error (MAPE). All the experimentally chosen combinations of the learning algorithm settings and other forecasting approaches were compared using a Root mean squared error (RMSE).

Two datasets (2012, 2013), which contain hourly measured values of the atmospheric pressure, were used in this experiment. The dataset from 2012 was selected for NN training. The trained NN was tested using the dataset from 2013. The prediction of the solar energy availability for the following day based on the test dataset (2013), the

calculation of the evaluation parameters (MAPE, RMSE), and the storage of the forecast output data is intended. The evaluation parameters were compared among all chosen combinations of the learning algorithm settings.

A. Deep Learning Experimental Settings

A deep feedforward network approach was selected for the realization of the experiment. The structure of the network was designed by an experimentally chosen number of layers and various types of activation functions (sigmoid, hyperbolic tangent, softsign, and ReLU (R), identity (I)). BP in three variants of gradient mode (mini-batch gradient descent (M), stochastic gradient descent (S), and batch gradient descent) was selected for the network training. BP in mini-batch mode was used to learn a model using back-normalization (BN). Due to the small number of data samples, the batch size was always chosen for the mini-batch gradient descent variant in the range from 0 to 10. The optimization was chosen among three optimization modes (ADAM, RMSProp with Nesterov Momentum (RMSProp) and Nesterov Momentum (NEST), none of them). There were three options to select from the initialization of the weights (Xavier, Sigmoid, and Glorot). The network configuration and training process settings were chosen experimentally.

B. Experimental Data

The experiment input data consisted of hourly measured values of the atmospheric pressure. The dataset from the entire year of 2012 was used for training the deep NN, and the dataset from the entire year of 2013 was used for the algorithm testing. The values of the atmospheric pressure in one day (24 values) and the value of available daily solar energy represented one input vector of the NN. The output vector consisted of an estimate of the value of the available solar energy for the following day. The range of input and output data values was normalized to a range from 0 to 1.

The input vector (the measured pressure values in *Pa*) and the output vector (the values of the daily solar energy availability in J/m^2) of the training dataset must be normalized. Therefore, a data scaling procedure by mean normalization (19) was implemented

$$\bar{z} = \frac{\bar{x} - \mu}{\max(x) - \min(x)}, \tag{19}$$

where $\bar{x}$ is the vector of original values, $\bar{z}$ is the vector of new values, μ is the average value, a and b are the min. and max. values of the required interval.

Equation 20 was used to calculate an estimate of the available daily solar energy $\hat{E}_A$ in J/m^2, which is dependent on the day of the year, solar elevation, and solar altitude

$$\hat{E}_A = 3600\tau G \int_{-\phi}^{\phi} \sin\theta. \tag{20}$$

These parameters are represented by: Φ is the site latitude, τ is the transmissivity estimate, and G is the extra-terrestrial solar radiation. This calculation was taken from the work in [30] and $\hat{E}_A$ values were used for designing the output vector of the training dataset for the NN learning.

IV. EXPERIMENTAL EVALUATION

A. Reference Methods

The ability of the deep learning system to estimate daily solar energy was compared to the evolutionary-fuzzy prediction scheme (FR) and support regression model (SV). The reference results, which stated FR demonstrates better results than SV, were obtained from the work in [30].

B. Evaluation Parameters

After the NN training, the NN was tested using a dataset from the year 2013. The results of the forecast algorithm were validated by calculation of MAPE (21)

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - y_i|}{y_i}, \tag{21}$$

where n is the number of the predictions, y_i is the real value of the pressure, and x_i is the predicted value of the pressure.

To compare the accuracy of the forecast with other approaches, the RMSE (22) was calculated

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{22}$$

where *n* is the number of the predictions, y_i is the real value of the pressure, and x_i is the predicted value of the pressure.

V. RESULTS AND DISCUSSION

The NN was tested with the experimentally chosen settings of the training parameters. The experiment used two datasets which contain hourly measured values of the atmospheric pressure. The first of them was used for network training and the second one was used for the trained network testing. The evaluation parameters were calculated to compare them to each other and to the results of the reference methods for each experiment configuration.

Table I shows the MAPE and RMSE values of the individual configuration settings in the deep learning experiment, which performed best in experimental testing. The selected methods of the individual configurations are described in Table II. The N2 configuration (Nesterov, Batch Normalization) has the lowest value of MAPE (0.235) of all the proposed setting combinations. A comparison of MAPE values of the N1 (0.244) and N2 (0.235) configurations, which had the same settings except for the number of layers, shows that it is better to choose a larger number of layers for a more accurate forecast. The A1 (MAPE = 0.249) and A2 (MAPE = 0.240) also had the same configuration apart from the initialization method and their comparison showed that the mode with Sigmoid is better than with Xavier. Once the optimization methods were

compared, it was shown that the best results are characterized by NEST (N1 - MAPE = 0.244), followed by ADAM (A1 - MAPE = 0.249, A4 - MAPE = 0.310) and RMSProp (R1 - MAPE = 0.330).

It was also observed that the vanishing-exploring gradient problem occurs in several initial iterations with inappropriately chosen settings. The vanishing-exploring gradient problem more often occurs at settings without an optimization method or, in the case of the NEST optimization method, with an inappropriately chosen learning coefficient, which confirms the described theoretical background. The RMSProp and ADAM optimization methods generally required a smaller value of the learning coefficient then NEST. The learning coefficient value was also dependent on the selected learning mode, i.e., if a stochastic gradient descent or mini-batch gradient descent was selected (for the mini-batch gradient descent, it also depended on the batch size). In general, it could be said that the result of the learning was highly conditioned by the selection of the learning coefficient. The speed of the convergence was also affected by the initial parameter initializations, which were initialized by the Xavier or Sigmoid method. Better results were obtained with the Xavier initialization method.

During the testing of various testing configurations, it was observed that if the total iteration number was increased, the network was re-learned and was not able to respond to sharp fluctuations (typical for the summer period). The forecast of the re-learned network corresponded to a filtering type or actual value averaging. The optimal number of iterations was between 10,000 and 15,000 for the value of MAPE of each configuration except A3 and A4 (the number of iterations was set at 80,000).

Regarding the gradient mode, it was empirically established that the batch gradient was not suitable for network training with the dataset (2012) because of unsatisfactory results. Therefore, no configuration with the batch gradient mode appeared in the top results of Table I and the gradient descent mode was just chosen from among the mini-batch mode and stochastic batch mode. A comparison of the results of A3 (MAPE = 0.240) with A1 (MAPE = 0.249) and A4 (MAPE = 0.310) shows that the stochastic gradient descent is significantly better than the mini-batch.

A comparison of the results from the previous study [4] to the deep learning experiment is shown in Table III. The table shows that the best result was achieved using the evolutionary fuzzy rules model (FR), which exceeds all the predictions obtained. However, the best forecasts from the trained NN surpassed all the above support vector regression models (SV). It could also be said that the forecasts in spring, autumn, and winter were more accurate in the case of the deep learning experiment.

Figure 2 shows that the NN presents satisfying predictions in early spring, autumn, and winter because there were no significant fluctuations in daily solar energy availability. In terms of summer months and late spring, the forecast did not correspond to sudden fluctuations, which are typical for these months. In Fig. 2, a prediction of the specific configurations (A2, N2, and R2) is shown. The differences among these individual configurations are not statistically significant, as evidenced by the MAPE values.

TABLE I. RESULTS OF THE EXPERIMENTAL INDIVIDUAL CONFIGURATION SETTINGS.

Label	MAPE	RMSE $(\mathbf{MJ/m^2})$
A1	0.249	2.958
A2	0.240	2.945
A3	0.240	3.009
A4	0.310	3.248
N1	0.244	3.020
N2	0.235	2.935
R1	0.330	3.402
R2	0.249	2.975

TABLE II. PARTICULAR COMBINATIONS OF THE INDIVIDUAL SETTING CONFIGURATIONS.

Label	Algorithm	Mode	Optimalization	Initiation	Act. function
A1	BP	M	ADAM	Xavier	R, R, I
A2	BP	M	ADAM	Sigmoid	R, R, I
A3	BP	S	ADAM	Xavier	R, R, R, I
A4	BN	M	ADAM	Xavier	R, R, R, I
N1	BN	M	NEST	Xavier	R, R, I
N2	BN	M	NEST	Xavier	R, R, R, R, R, I
R1	BP	M	RMSProp	Xavier	R, R, I
R2	BP	M	ADAM	Sigmoid	R, R, R, R, R, I

TABLE III. COMPARISON OF THE DEEP LEARNING EXPERIMENT RESULTS (N2) TO REFERENCE APPROACHES (FR, SV R_L, SV_RP, SV_RR).

Model	MAPE	RMSE
N2	0.235	2.935
FR	0.224	2.960
SV R_L	0.267	3.160
SV RP	0.362	3.444
SV RR	0.387	3.472

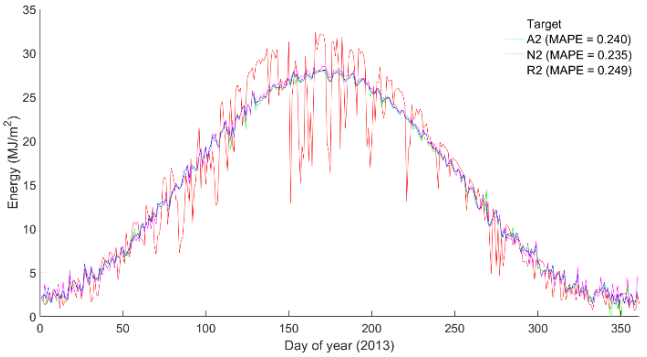

Fig. 2. Comparison of the solar energy prediction for various configurations of the proposed deep learning approach.

VI. CONCLUSIONS

This paper introduces the application of deep learning methods with the aim of training a multilayer NN, which can use a prediction tool to estimate future incoming solar energy. The BP algorithm in combination with deep learning methods, such as optimalization, batch normalization, and the initialization of weights, was used for network training.

In the results, it was observed that the N2 configuration had the lowest value of MAPE (0.235) than the other deep learning settings configurations. This MAPE value was 0.032 less than the lowest value of the support vector regression method (the MAPE value of the SV R_L was 0.267). It could be said that the results of the experiment are

comparable, and settings of the proposed solution are better than in models using the support vector regression method because all the MAPE values of the support vector regression are higher than the MAPE values of the proposed algorithm (N2). The results also showed that the forecast of the deep learning model did not perform with as satisfactory results in comparison to the evolutionary fuzzy rules method (the MAPE value of the FR was 0.224), based on that the MAPE value of N2 was 0.011 greater than of FR. Also, it could be clearly concluded that the NN gave satisfying predictions in early spring, autumn, and winter because there were no significant fluctuations in the daily solar energy availability. In terms of the summer months and late spring, the forecast did not correspond to sudden fluctuations, which are typical for these months.

In the article in [31], there were used six neural methods to predict energy consumption - artificial neural networks (ANN), general regression trees (CART), exhaustive regression trees (CHAID), support regression trees (SRT), support vectors (SV), and multivariant method adaptive regression splines (MARS). The MAPE results ranged from 0.160 to 0.350, ANN (MAPE = 0,225 - 0,280), CART (MAPE = 0,190 - 0,280), CHAID (MAPE = 0,230 - 0,280), MARS (MAPE = 0,175 - 0,350), SRT (MAPE = 0,165 - 0,215), and SV (MAPE = 0,205 - 0,335). It could be said that the results of the proposed deep leaning experiment are comparable (MAPE of N2 = 0.235).

The time and computational complexity for NN training was considerable, and therefore it was assumed the network was trained on an external computer or a cloud, where only the network parameters have been obtained and transferred to the embedded devices. Using the parameters obtained in this way, it was simple to implement a prediction model of an NN on an embedded system, because in the prediction phase, the network is represented by a look-up table. The memory consumption of the model depended on the size of the network (which determines the total number of parameters representing the NN), the type of architecture and the selected data type.

It is important to emphasize that the outcome of the learning process was greatly affected by the initialization of the parameters that used pseudo-random number generators. Therefore, different NN training results could be achieved with the same settings. A possible improvement could be the use of more datasets measured over several years to train the NN. Another possible aspect could be using the Dropout or L2 control method.

ACKNOWLEDGMENT

The authors would like to thank Jan Hájek from VSB - Technical University of Ostrava for cooperation during the simulation tests and implementation of the deep learning procedures.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] M. Cepenas, B. Peng, D. Andriukaitis, C. Ravikumar, V. Markevicius, N. Dubauskiene, D. Navikas, A. Valinevicius, M. Zilys, A. Merfeldas, and N. Hinov, “Research of PVDF Energy Harvester Cantilever Parameters for Experimental Model Realization,” *Electronics*, vol. 9, no. 12, p. 2030, Dec. 2020 DOI: 10.3390/electronics9122030

[2] P. Kromer, M. Prauzek, and P. Musilek, “Harvesting-aware control of wireless sensor nodes using fuzzy logic and differential evolution”, in *Proc. of 2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking Workshops, SECON Workshops 2014*, 2014, pp. 51–56. DOI: 10.1109/SECONW.2014.6979705.

[3] U. B. K. Ramesh, S. Sentilles, and I. Crnkovic, “Energy management in embedded systems: Towards a taxonomy”, in *Proc. of 2012 First International Workshop on Green and Sustainable Software (GREENS)*, 2012. DOI: 10.1109/GREENS.2012.6224254.

[4] A. Alzahrani, P. Shamsi, C. Dagli, and M. Ferdowsi, “Solar irradiance forecasting using deep neural networks”, *Procedia Computer Science*, vol. 114, pp. 304–313, 2017. DOI: 10.1016/j.procs.2017.09.045.

[5] A. Alzahrani, J. W. Kimball, and C. Dagli, “Predicting solar irradiance using time series neural networks”, *Procedia Computer Science*, vol. 36, pp. 623–628, 2014. DOI: 10.1016/j.procs.2014.09.065.

[6] C. Voyant, G. Notton, S. Kalogirou, M.-L. Nivet, C. Paoli, F. Motte, and A. Fouilloy, “Machine learning methods for solar radiation forecasting: A review”, *Renewable Energy*, vol. 105, pp. 569–582, 2017. DOI: 10.1016/j.renene.2016.12.095.

[7] J. Martens and I. Sutskever, “Training deep and recurrent networks with hessian-free optimization”, in *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science*, vol. 7700. Springer, Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-35289-8_27.

[8] E. B. Ssekulima, M. B. Anwar, A. Al Hinai, and M. S. El Moursi, “Wind speed and solar irradiance forecasting techniques for enhanced renewable energy integration with the grid: A review”, *IET Renewable Power Generation*, vol. 10, no. 7, pp. 885–898, 2016. DOI: 10.1049/iet-rpg.2015.0477.

[9] D. Díaz-Vico, A. Torres-Barrán, A. Omari, and J. R. Dorronsoro, “Deep neural networks for wind and solar energy prediction”, *Neural Processing Letters*, vol. 46, no. 3, pp. 829–844, 2017. DOI: 10.1007/s11063-017-9613-7.

[10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning”, *Nature*, vol. 521, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[11] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, “Deep neural networks for acoustic modeling in speech recognition”, *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012. DOI: 10.1109/MSP.2012.2205597.

[12] D. Andriukaitis, A. Laucka, A. Valinevicius, M. Zilys, V. Markevicius, D. Navikas, R. Sotner, J. Petrzela, J. Jerabek, N. Herencsar, D. Klimenta, “Research of the Operator’s Advisory System Based on Fuzzy Logic for Pelletizing Equipment,” *Symmetry*, vol. 11, no. 11, p. 1396, Nov. 2019 DOI: http://dx.doi.org/10.3390/sym11111396

[13] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, “Sample size selection in optimization methods for machine learning”, *Mathematical Programming*, vol. 134, no. 1, pp. 127–155, 2012. DOI: 10.1007/s10107-012-0572-5.

[14] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures”, in *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science*, vol. 7700. Springer, Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-35289-8-26.

[15] L. Jing, T. Wang, M. Zhao, and P. Wang, “An adaptive multi-sensor data fusion method based on deep convolutional neural networks for fault diagnosis of planetary gearbox”, *Sensors*, vol. 17, no. 2, p. 414, 2017. DOI: 10.3390/s17020414.

[16] N. M. Nawi, F. Hamzah, N. A. Hamid, M. Z. Rehman, M. Aamir, and A. R. Azhar, “An optimized back propagation learning algorithm with adaptive learning rate”, *International Journal on Advanced Science, Engineering and Information Technology*, vol. 7, no. 5, pp. 1693–1700, 2017. DOI: 10.18517/ijaseit.7.5.2972.

[17] M. Li, T. Zhang, Y. Chen, and A. J. Smola, “Efficient mini-batch training for stochastic optimization”, in *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 661–670. DOI: 10.1145/2623330.2623612.

[18] Z. Zhao, Z. Yang, H. Lin, J. Wang, and S. Gao, “A protein-protein interaction extraction approach based on deep neural network”, *International Journal of Data Mining and Bioinformatics*, vol. 15, no. 2, pp. 145–164, 2016. DOI: 10.1504/IJDMB.2016.076534.

[19] J. Schmidhuber, “Deep learning in neural networks: An overview”, *Neural Networks*, vol. 61, pp. 85–117, 2015. DOI: 10.1016/j.neunet.2014.09.003.

[20] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (IndRNN): Building A longer and deeper RNN", in *Proc. of 2018 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5457–5466. DOI: 10.1109/CVPR.2018.00572.

[21] D. Xie, J. Xiong, and S. Pu, "All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation", in *Proc. of 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, pp. 5075–5084. DOI: 10.1109/CVPR.2017.539.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *CoRR*, 2015.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proc. of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, vol. 15 of JMLR: W&CP 15, pp. 315–323.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proc. of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, vol. 9 of JMLR: W&CP 9, pp. 249–256.

[25] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Cham: Springer, 2018. DOI: 10.1007/978-3-319-94463-0.

[26] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks", in *Proc. of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

[27] S. Ioffe and Ch. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in *Proc. of the 32nd International Conference on Machine Learning*, 2015, vol. 37, pp. 448–456.

[28] D. Arpit, Y. Zhou, B. U. Kota, and V. Govindaraju, "Normalization propagation: A parametric technique for removing internal covariate shift in deep networks", in *Proc. of the 33rd International Conference on Machine Learning, ICML 2016*, 2016, pp. 1168–1176.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] P. Kromer, P. Musilek, J. Rodway, M. Reformat, and M. Prauzek, "Estimating harvestable solar energy from atmospheric pressure using support vector regression", in *Proc. of 2015 International Conference on Intelligent Networking and Collaborative Systems*, 2015, pp. 192–199. DOI: 10.1109/INCoS.2015.58.

[31] T. Szul, K. Nęcka, and T. G. Mathia, "Neural methods comparison for prediction of heating energy based on few hundreds enhanced buildings in four season's climate", *Energies*, vol. 13, no. 20, p. 5453, 2020. DOI: 10.3390/en13205453.