Energy Consumption Prediction through Deep Neuroevolution

Alexander Soudaei, Mohamed Elmi, Mikael Tsechoev, Zishan Khan, Ahmed Abbas, Jianhua Zhang

***Faculty of Technology, Arts and Design, Oslo Metropolitan University, 0166 Oslo, Norway*

E-mails: jianhuaz@oslomet.no, asoudaei@gmail.com

Abstract:
In this work we propose a deep learning model to predict the energy consumption of a low-energy house. In the deep learning model constructed, we use a differential evolution algorithm to automatically determine the best architecture of the deep neural network model. The energy prediction results are presented and analyzed to show the effectiveness of the deep neuroevolution scheme developed.

Keywords – Deep learning; Differential evolution; Swarm intelligence; Evolutionary algorithm; Energy prediction; Regression model.

1. INTRODUCTION

In this paper, we present a new solution to predict the energy use of a low-energy house. This solution employs an evolutionary algorithm, called Differential Evolution (DE), to optimize the architecture and training of deep learning (DL) models.

DE was inspired by the evolutionary aspects of living organisms. The evolutionary plausibility makes the DE a competitive candidate for selecting the optimal architecture of a deep neural network, since it provides remarkable approximate solutions that cannot be easily obtained by using other algorithms. Approximate solutions are also a good way to mitigate the need for exact solutions since the availability of sufficient computational power may hinder the normal user of most of the devices today from developing their DL solutions.

1.1 LITERATURE REVIEW

Slowik and Bialko (2008) presented a solution using DE to optimize ANNs for classification of parity-p problem. The results of the proposed solution were compared to the results of other algorithms such as the Levenberg-Marquardt method and backpropagation. DE-ANN had better results than the other algorithms, where DE-ANN had a higher percentage of correctly classified data in four of ten possible cases compared to the results of the LM algorithm. This work inspired us to apply this combination of solutions, DE+ANN, to another type of problem that we presented in this paper, a regression problem.

It is presented 4 different solutions for energy prediction in appliances in Candanedo et.al. (2007) Multiple linear regression, (2) Gradient Boosting Machines (GBM), (3) Support Vector Machine (SVM) with radial kernel, and lastly (4) Random Forest. Where GBM has achieved the best results by looking at the RMSE and R^2. These algorithms are all stand-alone algorithms that are only within the confines of their abilities. This inspired our work for presenting a solution with better results by using a combination of algorithms.

One of the most similar works to ours is presented by *Pierre et.al (2021)* for Artificial Neural Network optimized by differential evolution for predicting diameters of jet grouted columns. Though the results displayed in the paper shown to be better, it is not guaranteed that these results are in fact better. Using only R^2 as a metric for accuracy is not enough, and is often coupled with RMSE, MAE and MAPE. For our work, we use all four mentioned metrics to get a profound sense of the accuracy of our work and to compare it better. Hancer (2020) has also used different detailed ways of presenting the results. This shows that although metrics may be different, they have to be complete.

ANN is not the only type of neural networks that can be coupled with DE or other EAs. There has been a lot of interesting research when it comes to other types of neural networks combined with other algorithms to make predictions. In Mermarzadeh and Keynia (2021), a solution has been presented for short-term prediction in electricity load and price by combining neural networks with LSTM.

On the other hand, other EAs can also be coupled with a Neural Network. In Hong et.al(2020), it presented a genetic algorithm (GA) combined with Convolutional Neural Network (CNN) to optimize hyperparameters where the verification time and accuracy have been evaluated with a fitness evaluation. The algorithm in general showed promising results and reduced the training time.

2. PROBLEM STATEMENT AND DATASET DESCRIPTION

The dataset used for developing this algorithm is a time-series dataset. It has 28 features, one label and 19736 instances, where the 28 features measure the humidity and temperatures of 9 different rooms in a house, wind speed, pressure, and dew point among others with a Zigbee WSN.

Appliances is the target variable, also referred to as label we are trying to predict, measuring the energy consumption rate of appliances in the house in watts per hour (Wh).

The datapoints are measured every 10th minute from January 11th 2016 to may 27th 2016.

There are no values missing in this dataset. you can look at Candanedo et.al. (2007) for more information about the dataset.

When working with this dataset there is a feature that must be dealt with before using it to train our model. Given the nature of the dataset being a time-series, the date-feature of the dataset is not in neither integer- nor a float-format.

This poses a problem since a string-format cannot be easily converted to an integer or a float. The solution for this was to divide the date-feature into 3 separate features called Day, Month, and Time. After this conversion we now have a total of 30 features in our dataset.

Another typical problem to deal with are Outliers. Outliers are extreme data points that deviate the normal intervals of the rest of the data points.

To gain more understanding of the distribution of the variables in the dataset, we used histograms to visualize them in Fig. 1.

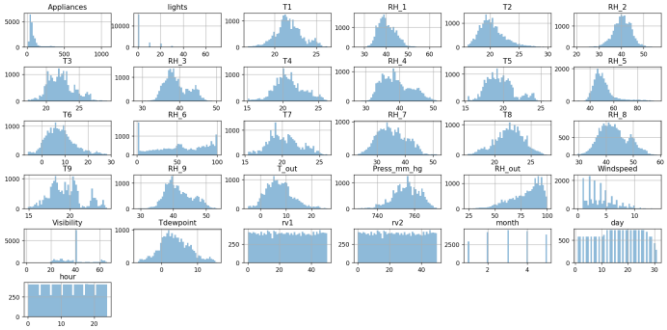

Fig. 1. Histogram of each of the 31 input variables (features).

Looking at the distribution of the different values, we can see that most of the variables are evenly distributed, while a few of them stand out.

The appliances column seems so be slightly right-skewered, which is something important to consider as this is the column we are trying to predict.
Lights, month, day and hour also stands out here, as they are be a discrete variables which only take on certain values.

Detecting and removing outliers is a common practice in machine learning, as this allows us to focus on and get better results on the "regular" day-to-day values, than if we included the outliers.

We remove these outliers (extreme values) to get better results of the day-to-day values. This makes sense as we would rather want a better accuracy on values that we commonly see, than to get a decent accuracy on both the common and extreme values.

To achieve this, we will use the 3-sigma method. This means that instances where the appliance value is less than mean-3*std or higher than mean+3*std, will be removed from the dataset prior to machine learning.

After this process, 540 instances have been removed from the dataset and 19195 instances remain.

Fig. 2 shows the correlation between the different variables in the dataset, from which we can see that the four most correlated features to appliances are Lights, T2, and T6 with a positive correlation and RH_out with a negative correlation.

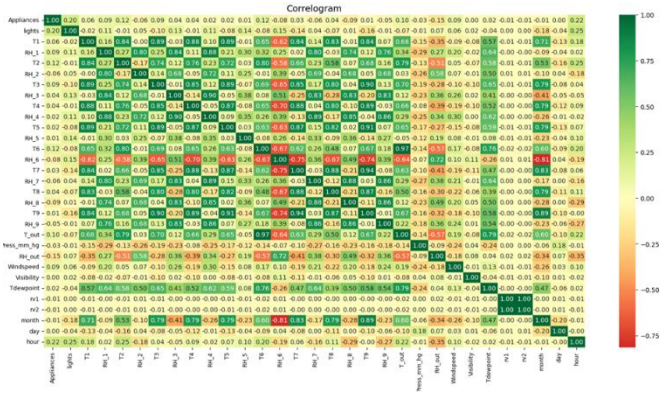

Fig. 2. Correlation matrix between the features (containing both positive and negative correlations).

3. DEEP NEUROEVOLUTION APPROACH BY COMBINING DE WITH NEURAL NETWORK

Differential evolution (DE) is a heuristic, stochastic, population-based algorithm. Some of the many advantages of this algorithm is that it has fast convergence and finding global minimum. DE uses evolutionary principles like Genetic algorithm (GA), to alter the behavior of the swarm. These are Mutation and Crossover, but are performed differently than GA. Bilal et.al (2020) in detail, presented all the different evolutions of the DE-algorithm throughout the year since 1995. In Storn and Price (1997), mutation is referred to the generation of a new position in the search space from three vector donors. The mutation can be created by:

$$v = v_1 + F\left(v_2 - v_3\right), \text{ where } F \in [0,2] \quad (1)$$

After the mutation is created, the crossover operator is applied to create candidate vector $\boldsymbol{u}$, where the components of $\boldsymbol{u}$ are selected one-by-one as follows:

$$u_i = \begin{cases} v_i, & \text{if } r < CR \text{ or } i = I \\ x_i, & \text{otherwise} \end{cases} \quad (2)$$

where u_i is the i-th component of the candidate vector, r is a random number in the unit interval [0, 1], CR stands for the Crossover Probability, I is a random number in the intervals of [0, n dimensions] and x is the position of a particle. Both the DE-style mutation and Crossover must be done to generate a new position. This step-by-step process eliminates

a higher degree of random position generation, which has been used by many other optimization algorithms.

In this work we combine the DE algorithm and backpropagation to optimize ANNs. The DE will be used for optimizing the number of nodes in the hidden layers of an ANN with a predetermined number of hidden layers.

3.1 Encoding scheme

We represent ANN architecture as a position vector where the first component represents the number of nodes in the first hidden layer, the second component represents the number of nodes in the second hidden layer, and so on. The input and output layers are not included as this is predetermined and constant.

Here is an example of an ANN with 51 nodes in the first hidden layer and 33 nodes in the second hidden layer, represented as a position vector [51 33].

The positions are then normalized to numbers between 0 and 1, with the goal of faster swarm convergence.

3.2 Method

As described in 3.1, we represent ANN architectures as position vectors. Every architecture is evaluated by first training it with SGD with the goal of fast yet quality convergence. Then we put the architecture through a second training phase using Adadelta optimizer which has an adaptive learning rate, is very slow but has proven to give quality convergence. This two-phase training process is repeated twice for every architecture and we use the best result from the two trials is used as its final measure of its quality, due to random weight and bias initializations in the ANN's allowing the same architecture to produce different results in different trials. Ideally we would run more than two trials per architecture however we were limited by computational power. After evaluation, positions are tupdated using DE.

3.3 Parameters used in the experiment

Table 1. The setting of parameters used in the experiment.

Parameter	Description	Value
npart	Number of particles in the swarm	20
ndim	Number of dimensions in the search space of the swarm	4
m	Max number of swarm generations	30
CR	Crossover rate	0.5
F	Mutation rate	0.8
minnodes	Minimum number of nodes in a hidden layer	30
maxnodes	Maximum number of nodes in a hidden layer	80
n_inputs	Number of input nodes	30
n_outputs	Number of output nodes	1
nhidden	Number of hidden layers	2
batch_size	Number of datapoints to use per parameter update	200
epochs	Number of times the whole dataset is revised by the neural network during the training process	1000
patience	Number of epochs without improvement before stopping the training process of a	50
optimizer	Optimization algorithm used for updating the parameters in the neural network	SGD and Adadelta
lr	Learning rate, how much we take a step in the proposed direction of the optimizer to update ANN parameters	0.001, decreased by 25% if there is no improvement for the last 30 epochs
Hidden layer activation function	Activation function used in the hidden layer nodes	SoftPlus
Output layer activation function	Activation function used in the output layer node	ReLu
Loss function	The function we use to evaluate the fitness of a particle	MSE (Mean squared error)

4. RESULTS

Table 2. The results of DE search.

Best validation MSE in generation 0	2369.34
Best architecture found	30-36-33-53-46-1
Best architecture validation MSE	2271.02
Generation in which the best architecture was found	17
Swarm MSE reduction	98.32

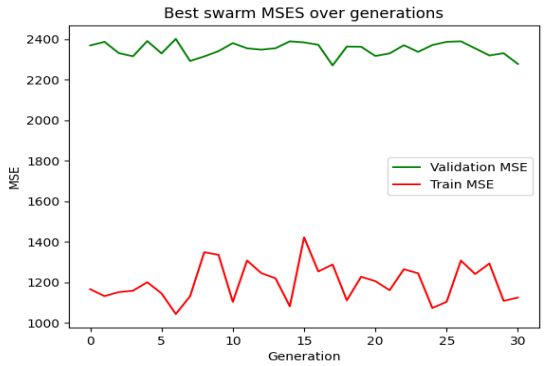

Fig. 3. The change of the best training- and validation-MSE in the swarm across 30 generations.

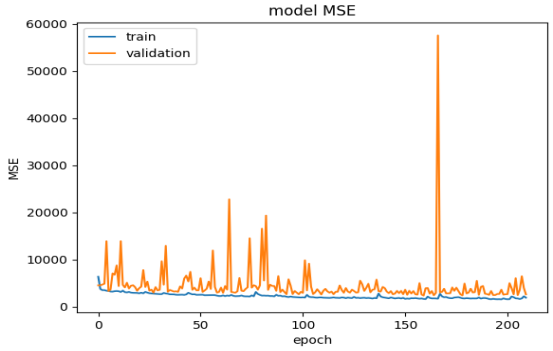

Fig. 4. The training- and validation MSE over 230 epochs of the initial training of the NN with the best architecture (SGD optimizer used for network training).

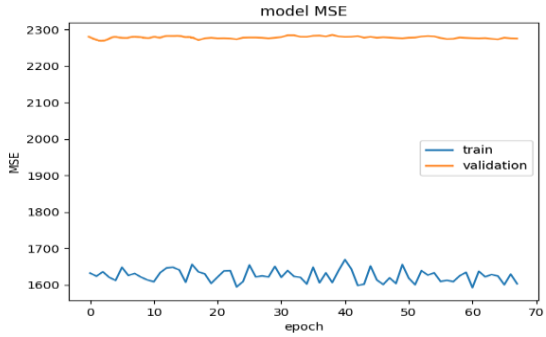

Fig. 5. The training and validation MSE over 67 epochs in the retraining of the NN with the best architecture.

Table 3. Training, validation, and test results of the

DE-ANN model constructed.

Training	RMSE	38.49
	R^2	0.68
	MAE	21.61
	MAPE	26.80
Validation	RMSE	47.65
	R^2	0.48
	MAE	25.88
	MAPE	32.34
Testing	RMSE	52.79
	R^2	0.38
	MAE	27.63
	MAPE	32.66

Table 4. Comparison of results of different algorithms (c.f., Table 5 in Candanedo, Feldheim and Deramaix (2017)).

Model	Parameters/features	Training				Testing			
		RMSE	R^2	MAE	MAPE %	RMSE	R^2	MAE	MAPE %
LM	Light, T1,RH1,T2,RH2,T3, RH3,T4, RH4,T5,RH5,T6, RH6, T7,RH7,T8,TH8,T9,RH9, To,Pressure,Rho,Windspd, Tdewpoint, NSM, WeekStatus, Day of Week	93.21	0.18	53.13	61.32	93.18	0.16	51.97	59.93
SVM Radial	Light,T1,RH1,T2,RH2,T3,RH3, T4,RH4,T5,RH5,T6,RH6,T7,RH7,T8,TH8,T9,RH9,To, Pressure,Rho,WindSpeed, Tdewpoint,NSM, WeekStatus, Day of Week	39.35	0.85	15.08	15.60	70.74	0.52	31.36	29.76
GBM	Light,T1,RH1,T2,RH2,T3,RH3, T4,RH4,T5,RH5,T6,RH6, T7,RH7,T8,TH8,T9,RH9,To, Pressure,Rho,WindSpeed, Tdewpoint,NSM, WeekStatus, Day of Week	17.56	0.97	11.97	16.27	66.65	0.57	35.22	38.29
RF	Light,T1,RH1,T2,RH2,T3,RH3, T4,RH4,T5,RH5,T6,RH6,T7, RH7,T8,TH8,T9,RH9,To, Pressure,Rho,WindSpeed, Tdewpoint,NSM, WeekStatus, Day of Week	29.61	0.92	13.75	13.43	68.48	0.54	31.85	31.39

In Candanedo, L.M., Feldheim, V., Deramaix, D. (2017), Data driven prediction models of energy use of appliances in a low-energy house, *Energy and Buildings*, Volume 140, 81-97 the researchers concluded that GBM was the best model.

Looking at the test results from our approach and theirs, we can see that our neuroevolutionary approach beat their GBM results on three out of four metrics except for r squared.

5. STATISTICAL ANALYSIS OF MODEL TESTING ERRORS

Getting a deeper understanding of the testing error is an important step to get practical knowledge about the model. Without digging into it, we only have the metrics to tell us how it really is performing. The metrics do not give us a clear enough image of which instances we have good accuracy on and which ones we do not. The table below shows statistical error measurements to gain a more profound understanding of the inner workings of the proposed model.

Table 5. The statistics of the testing errors defined by the actual value minus the predicted value on every instance in the test set.

Standard Deviation	52.69
Max	333.22
75th percentile	10.09
Mean	3.23
25th percentile	-14.87
Min	-278.82
Proportion of negative errors	0.55
Proportion of positive errors	0.45

Notice that the model usually predicts values too high, but the mean error being positive means that the magnitude of too low predictions outweighs the magnitude of too high predictions. The max error is 332 and the minimum error is -249. This means that the worst error was when a prediction was lower than the actual value.

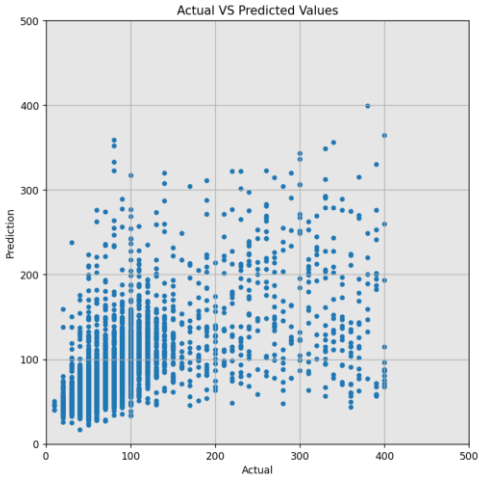

Fig. 6. The predicted vs. actual energy use.

A perfect model with zero error would show us points following a linear relationship with a slope of 1. When the actual values are in the lower range of approximately 10-150, the slope looks relatively good. However, values above 150 are clearly harder to predict for the model, where it frequently predicts too low values.

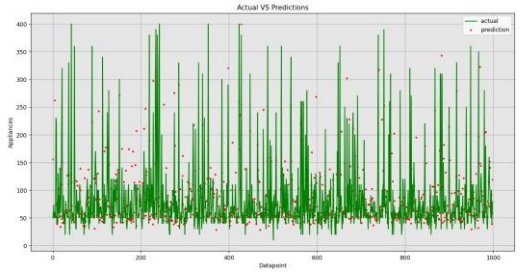

Fig. 7. The predicted vs. actual energy use (only the first 1000 instances in test dataset shown here as a sample).

Fig. 7 confirms that we have better accuracy when the actual values are in the lower range. When the actual values are in the higher range, the model usually predicts values much lower than the actual values. We can clearly see that when the actual values are higher than 150, the model has more difficulty making accurate predictions.

Fig. 8 shows that the majority of errors are negative (too high predictions), however we have many positive errors with greater magnitude.

Table 6 shows comparison of errors from too high predictions vs errors from too low predictions (in comparison with the actual values). In the table we use the absolute value of the testing errors, even though too high predictions result in negative errors and too low predictions lead to positive errors.

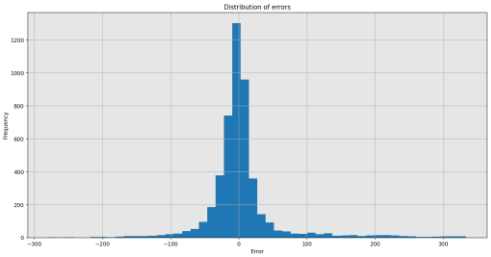

Fig. 8. The distribution of the testing errors.

Table 6. The statistics of the absolute values of the errors due to too low and too high predictions.

	Absolute errors due to too high predictions	Absolute errors due to too low predictions
Standard Deviation	29.00	58.13
Max	278.02	333.22
75th percentile	26.72	29.00
Mean	22.23	34.20
25th percentile	5.56	5.38
Min	0.006	0.01

Notice that 75% of the too high predictions are between 0.006 to 26.72 different from the actual value, while 75% of too low predictions are between 0.01 and 29.00 different from the actual value.

To sum it all up, the test error analysis shows that the model more frequently predicts values higher than the actual value, however this is outweighed by the magnitude of the error made when predicting values lower than the actual value. When the actual value is in the higher range, the model has difficulty predicting it accurately.

6. CONCLUSIONS

In this work, we present a new approach to finding the optimal structure of deep neural network for predicting the energy consumption of a low-energy house. The DE algorithm is used to determine the optimal architecture of deep network model. The results showed that our proposed method can reduce the testing MSE by 98.32 from the initial architecture to the best architecture found by DE algorithm (with a final MSE of 2271.02).

7. REFERENCES

Candanedo, L.M., Feldheim, V., Deramaix, D. (2017), Data driven prediction models of energy use of appliances in a low-energy house, *Energy and Buildings*, Volume 140, 81-97.

Slowik, A. and Bialko, M. (2008), Training of Artificial Neural Networks Using Differential Evolution Algorithms, *2008 Conference on Human System Interaction HSI 2008*, 60-65.

Storn, R., and Price, K.V. (1997), Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, *Journal of global optimization*, 341–359.

Hancer, E. (2020), A new multi-objective differential evolution approach for simultaneous clustering and feature selection, *Engineering Applications of Artificial Intelligence*, Volume 87, 103307.

Atangana Njock, P.G., Shen, S., Zhou, A., Modoni, G. (2021),Artificial neural network optimized by differential evolution for predicting diameters of jet grouted column, *Journal of Rock Mechanics and Geotechnical Engineering*, Volume 13, 1500 - 1512.

Bilal, Pant, M., Zaheer, H., Garcia-Hernandez, L., Abraham, A. (2020), Differential Evolution: A review of more than two decades of research, *Engineering Applications of Artificial Intelligence*, Volume 90, 1-24.

Choi, D., Han, Ji., Park, S., Hong, S. (2020), Hyperparameter Optimization Using a Genetic Algorithm Considering Verification Time in a Convolutional Neural Network, *Journal of Electrical Engineering & Technology*, Volume 15, 721 - 726.

Memarzadeh, G., and Keynia, F. (2020), Short-term electricity load and price forecasting by a new optimal LSTM-NN based prediction algorithm, *Electric Power Systems Research*, Volume 192, 1 - 14.

Ling, S., Li, H., Zhang, Miao., Pan, S., Su, S. (2021), Convolutional Neural Networks-Based Lung Nodule Classification: A Surrogate-Assisted Evolutionary Algorithm For Hyperparameters Optimization, *IEEE Transactions on Evolutionary Computation*, Volume 25, 869 - 882.

Bulac, C., Boicea, V.A., Picioroaga I.I., Sidea D.O., Tudose A.M. (2020), A CNN Based Model For Short-Term Load Forecasting: A Real Case Study on the Romanian Power System, *2020 55th International Universities Power Engineering Conference (UPEC)* , 1-6.