BULGARIAN ACADEMY OF SCIENCES
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS • 2021 • Vol. **75**, pp. 5-14
p-ISSN: 2738-7356; e-ISSN: 2738-7364
https://doi.org/10.7546/PECR.75.21.02

Multilayer Perceptron Fast Prototyping with Differential Evolution and Particle Swarm Optimization in LibreOffice Calc

Petar Tomov

Institute of Information and Communication Technologies
at the Bulgarian Academy of Sciences
Acad. Georgi Bonchev Str., block 2, office 514, 1113 Sofia, Bulgaria
petar.tomov@iict.bas.bg

Abstract: The multilayer perceptron is one of the most used kinds of artificial neural networks. It has an organization in layers and in most cases, these layers are fully connected. The training of multilayer perceptron is a process of finding optimal values for the weights between the layers. When optimal values are desired, LibreOffice Calc has a module called Solver. For non-linear optimization, the Solver module uses a hybrid algorithm which is a combination of differential evolution and particle swarm optimization. For using LibreOffice Calc in multilayer perceptron training the model of the artificial neural network should be deployed into the working sheet. This paper proposes fast prototyping by multilayer perceptron model deployment in such a way in which differential evolution and particle swarm optimization optimizers to be used in the training process.

Keywords: *differential evolution, multilayer perceptron, particle swarm optimization*.

1. Introduction

Financial time series forecasting has great importance in the modern world and the area of soft computing [1]. Many important decisions [2] are taken after analysis of the possible future changes in the financial instruments. When it comes to non-linear forecasting, artificial neural networks are one of the most discussed

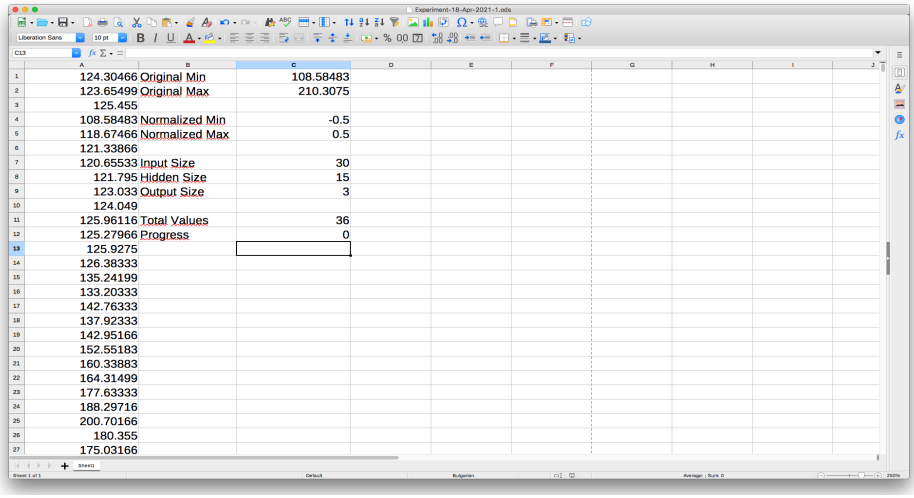
approaches [3]. The autocorrelation approach in artificial neural networks forecasting splits the time series into a conditional past window and a conditional future window [4]. After normalization [5] (in most cases linear normalization), the conditional past window feeds the artificial neural network input. The normalized values [5] of the conditional future window are expected at the output of the artificial neural network. The key point of the artificial neural networks used as a forecasting tool is their training [6]. In the case of multilayer perceptron, training of the artificial neural network consists of finding optimal values for the weights in the network in such a way that the network has generalization (predicting) capabilities. For achieving such generalization capabilities, the input-output examples are divided into three subsets: training set; validation set; testing set [7]. The training set is used in the process of optimal weight searching. The examples in the training set are used for weights values adjustment. The examples in the validation set are used for monitoring of generalization capabilities of the artificial neural network. These examples are not used for weights adjustment. These examples are used for training process stopping criteria when the generalization of the network drops down. The examples in the testing set are used as artificial neural network operational mode.

Before real industrial implementation (for example software as a service [8]) of an artificial neural network, the process of fast prototyping is very relevant. There are many software instruments for fast prototyping like Matlab, R, NeuroSolutions, and others, but fast prototyping also can be done very efficiently in LibreOffice Calc. For achieving such prototyping, the topology of the multilayer perceptron should be deployed in Calc's working sheet. The input layer usually has no calculating capabilities and it only repeats the input of the supplied example. Calculating formulas should be given for the hidden and output layer. Total network error should be calculated. This value is the exact objective for minimization. The process of minimization is done by changing a given range of cells which represent artificial neural network weights. Two common questions can be solved during fast prototyping in solving a particular forecasting problem: 1. How many hidden layers are needed; 2. How many neurons are needed in each hidden layer?

This paper proposes an application of the Solver module available in LibreOffice Calc, with its differential evolution and particle swarm optimizers, for artificial neural networks training.

2. Model deployment

Multilayer perceptron deployment in LibreOffice Calc sheet starts with feeling the most left column with values of the time series. Some parameters needed for the building of the model are given in the third column (Fig. 1).



	A	B	C
1	124.30466	Original Min	108.58483
2	123.65499	Original Max	210.3075
3	125.455		
4	108.58483	Normalized Min	-0.5
5	118.67466	Normalized Max	0.5
6	121.33866		
7	120.65533	Input Size	30
8	121.795	Hidden Size	15
9	123.033	Output Size	3
10	124.049		
11	125.96116	Total Values	36
12	125.27966	Progress	0
13	125.9275		
14	126.38333		
15	135.24199		
16	133.20333		
17	142.76333		
18	137.92333		
19	142.95166		
20	152.55183		
21	160.33883		
22	164.31499		
23	177.63333		
24	188.29716		
25	200.70166		
26	180.355		
27	175.03166		

Fig. 1. Given time series and model parameters

For the linear scaling of the original time series, the minimum and maximum values are found by Calc formulas $MIN(A:A)$ and $MAX(A:A)$. Input values sent to the artificial neural network are scaled in the specified range. The topology of the three-layer perceptron is also given. The counting of the total number of available values in the time series is done by Calc formula $COUNT(A:A)$. The size of the time series is needed because splitting the values in training examples is done according to this size. Model deployment is done with a specially written Python script [9]. Scripting languages can perform pretty slow and because of this, a separate Calc cell is devoted to keeping track of the model deployment progress.

Linear scaling of the time series values in column E is the first operation done by the script.

Python script for input scaling follows:

```
for t in range(1, total_values + 1):

sheet.getCellRangeByName("E"+str(t)).setValue(sheet.getCellR
angeByName("$C$4").getValue()+
    (sheet.getCellRangeByName("$C$5").getValue() -
sheet.getCellRangeByName("$C$4").getValue())*
    ((sheet.getCellRangeByName("A"+str(t)).getValue()-
sheet.getCellRangeByName("$C$1").getValue())/
    (sheet.getCellRangeByName("$C$2").getValue() -
sheet.getCellRangeByName("$C$1").getValue())))
```

Each layer has bias represented in the model as dummy neurons constantly emitting one as value.

Python script for biases setup follows:

```
sheet.getCellRangeByName("G" + str(x)).setValue(1)
sheet.getCellRangeByName("G" + str(x)).CellBackColor =
(255 << 16 | 255 << 8 | 0)
sheet.getCellRangeByName("H" + str(x)).setValue(1)
sheet.getCellRangeByName("H" + str(x)).CellBackColor =
(255 << 16 | 255 << 8 | 0)
sheet.getCellRangeByName("I" + str(x)).setValue(1)
sheet.getCellRangeByName("I" + str(x)).CellBackColor =
(255 << 16 | 255 << 8 | 0)
sheet.getCellRangeByName("J" + str(x)).setValue(1)
sheet.getCellRangeByName("J" + str(x)).CellBackColor =
(255 << 16 | 255 << 8 | 0)
```

For each training example, all three layers of the artificial neural network are formed (Fig. 2).

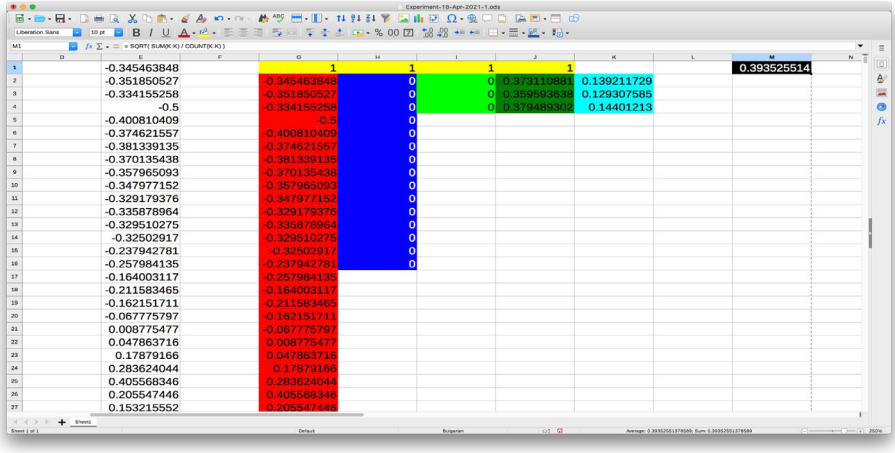


Fig. 2. Artificial neural network topology

The input layer only transfers the scaled values of the time series but for the hidden layer and the output layer incoming signals are multiplied by the weights and the sum is passed to the activation function. The expected values are taken just as they were scaled.

Python script for input data loading follows:

```
for i in range(1, input_size + 1):
    sheet.getCellRangeByName("G" + str(x +
i)).setValue(sheet.getCellRangeByName("E" +
    str(t + i)).getValue())
```

```
    sheet.getCellRangeByName("G" + str(x + i)).CellBackColor =
(255 << 16 | 0 << 8 | 0)
''' Expected data loading. '''
for e in range(1, output_size + 1):
    sheet.getCellRangeByName("J" + str(x +
e)).setValue(sheet.getCellRangeByName("E" +
    str(t + e + input_size)).getValue())
    sheet.getCellRangeByName("J" + str(x + e)).CellBackColor =
(0 << 16 | 127 << 8 | 0)

''' Setup hidden layer. '''
wih = 1
for h in range(1, hidden_size + 1):
    sum = ""
    for i in range(0, input_size + 1):
        sum = sum + "G" + str(x + i) + "*Q" + str(wih)
        wih = wih + 1
        if i < input_size:
            sum = sum + " + "
        sheet.getCellRangeByName("H" + str(x +
h)).setFormula("=TANH( " + sum + " )")
        sheet.getCellRangeByName("H" + str(x + h)).CellBackColor =
(0 << 16 | 0 << 8 | 255)

''' Setup output layer. '''
who = 1
for o in range(1, output_size + 1):
    sum = ""
    for h in range(0, hidden_size + 1):
        sum = sum + "H" + str(x + h) + "*S" + str(who)
        who = who + 1
        if h < hidden_size:
            sum = sum + " + "
    sheet.getCellRangeByName("I" + str(x + o)).setFormula
("=TANH( " + sum + " )")
    sheet.getCellRangeByName("I" + str(x + o)).CellBackColor =
(0 << 16 | 255 << 8 | 0)
```

The error for each training example is calculated between the artificial neural network output and the expected values. The total mean-square error of the artificial neural network is calculated by the usage of all training examples errors.

Python script for network output error and total network error follows:

```
for r in range(1, output_size + 1):
    sheet.getCellRangeByName("K" + str(x + r)).setFormula
("= (J" + str(x + r) + "-I" +
    str(x + r) + ") * (J" + str(x + r) + "-I" + str(x + r) + ")")
    sheet.getCellRangeByName("K" + str(x + r)).CellBackColor =
(0 << 16 | 255 << 8 | 255)
```

```
sheet.getCellRangeByName("M1").setFormula
("= SQRT( SUM(K:K) / COUNT(K:K) )")
sheet.getCellRangeByName("M1").CellBackColor = 0
```

For each training example network topology is reproduced but all reproductions are using the same cell regions given for the network weights. These weights are part of the operational replication of the artificial neural network topology (see Fig. 3).

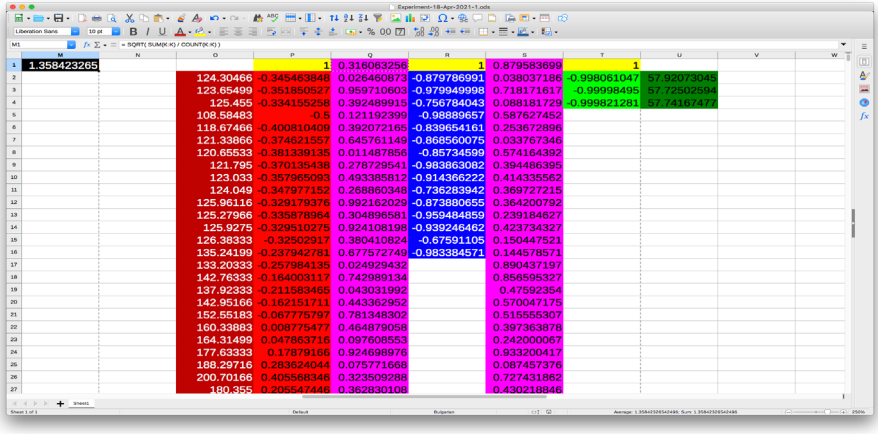


Fig. 3. Operational topology of the artificial neural network

Python script for weights setup follows:

```
wih = 1
for h in range(2, hidden_size + 2):
    for i in range(1, input_size + 2):
        sheet.getCellRangeByName("Q" + str(wih)).CellBackColor
= (255 << 16 | 0 << 8 | 255)
        wih = wih + 1

who = 1
for o in range(2, output_size + 2):
    for h in range(1, hidden_size + 2):
        sheet.getCellRangeByName("S" + str(who)).CellBackColor
= (255 << 16 | 0 << 8 | 255)
        who = who + 1

    sheet.getCellRangeByName("U" + str(o)).setFormula
("=$C$1 + ($C$2 - $C$1) *
    ((T" + str(o) + " - $C$4) / ($C$5 - $C$4))")
    sheet.getCellRangeByName("U" + str(o)).CellBackColor =
(0 << 16 | 127 << 8 | 0)
```

The artificial neural network is trained with the Solver module and when a new value appears in the time series the model is redeployed with the new value taken into account. Such continuous training is very important because most of the financial time series are not static but they are dynamic.

3. Experiments and results

All experiments are done on a single processor desktop machine - Intel Core i5, 2.3 GHz, 2 Cores, 8GB RAM with macOS High Sierra 10.13.6 and LibreOffice 7.0.5.2. As financial time series, publicly available rates of Bitcoin to USD is taken on a daily interval. For artificial neural network topology, 30 input nodes and 3 output nodes are chosen. It means that the past interval (the lag) is 30 days and the forecast interval (the lead) is 3 days. The size of the hidden layer is taken to be 15. It is almost half of the sum between input and output size. Hyperbolic tangent is chosen as an activation function for the hidden and the output layers. It is preferred because it is symmetric to the X-axis. The original time series values are scaled in the range of -0.5 to +0.5. Such scaling uses around half of the slope given by the hyperbolic tangent. All parameters of the DE-PSO optimizers are kept to their default values (Fig. 4).

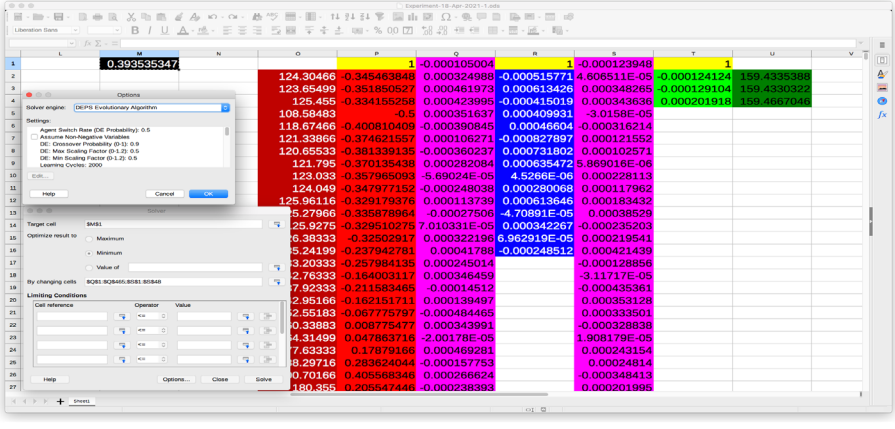


Fig. 4. Solver and optimizers parameters

Measurements of optimization performance are done on each 300 optimization cycle. On each 3000 optimizations cycle, a new value in the time series appears (Fig. 5). With each new time series value, the artificial neural network model in the LibreOffice Calc sheet is rebuilt. As it is shown in Fig. 5, the initial training does greater convergence. With each new value in the time series, convergence goes from a much lower value. Global non-linear optimization is very sensitive to the initial starting point.

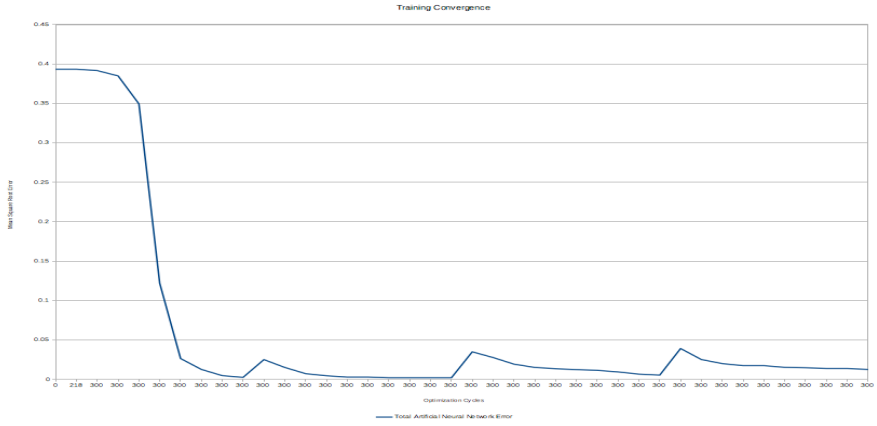


Fig. 5. Total artificial neural network error

In the experiments, 513 variables are subject to optimization (values of the weights). The initial values are randomly selected near to zero value. The training time goes up with the rise of the size in the time series as shown in Fig. 6.

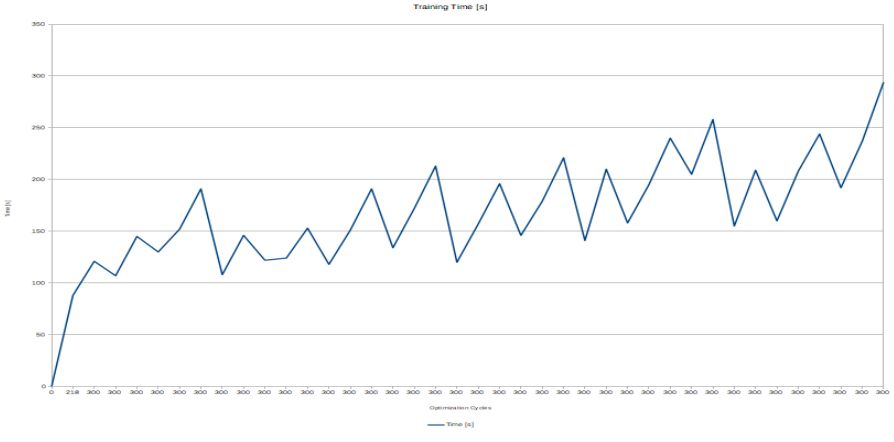


Fig. 6. Training time usage

It is logical because when the time series is bigger, more cells with formulas should be calculated. The experiments clearly show that the proposed prototyping is efficient.

4. Conclusion

In this paper, fast prototyping of artificial neural networks was proposed. Fast prototyping is done in LibreOffice Calc. The model of an artificial neural network is deployed into the LibreOffice Calc sheet with a specially written Python script.

Searching for optimal weights in the artificial neural network is done by the build-in Solver. The Solver uses differential evolution and particle swarm optimizers. The experiments proved that proposed fast prototyping is very promising and can be used in artificial neural network design.

As further research, artificial neural networks with a higher number of hidden layers to be implemented and hierarchical [10] training to be implemented. Human-computer interaction in modifying artificial neural networks weight is also an interesting aspect for research [11]. The Solver module in LibreOffice Calc has other optimizers different than DE-PSO. It will be interesting optimization to be done with them. LibreOffice Calc is open-source software that comes with some problems with quality. It is relevant additional quality validation [12] of the Solver to be done.

Acknowledgments

This research is funded by Velbazhd Software LLC and it is partially supported by the Bulgarian Ministry of Education and Science (contract D01–205/23.11.2018) under the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)", approved by DCM # 577/17.08.2018.

References

1. Angelova, V.: Investigations in the area of soft computing targeted state of the art report. Cybernetics and Information Technologies 9(1), 18-24 (2009).
2. Cvetkova, P., Pandulis, A., Borissova, D.: Application of information technologies to support mathematically reasoned decisions. In: Knowledge Society and 21st Century Humanism Proceedings, pp. 488-496, Academic Publishing House Za Bukvite - O Pismeneh, Sofia (2020).
3. Dhamija, A., Bhalla, V.: Financial time series forecasting: Comparison of neural networks and ARCH models. International Research Journal of Finance and Economics 49, 194-212 (2010).
4. Flores, J., Engel, P., Pinto, R.: Autocorrelation and partial autocorrelation functions to improve neural networks models on univariate time series forecasting. In: International Joint Conference on Neural Networks (IJCNN'12) Proceedings, pp. 1-8, IEEE, Rio de Janeiro, Brazil (2012).
5. Sola, J., Sevilla, J.: Importance of input data normalization for the application of neural networks to complex industrial problems. IEEE Transactions on Nuclear Science 44(3), 1464-1468 (1997).
6. Sideratos, G., Ikonomopoulos, A., Hatziargyriou, N.: A novel fuzzy-based ensemble model for load forecasting using hybrid deep neural networks. Electric Power Systems Research 178, 106025 (2020).

7. Kaastra, I., Boyd, M.: Designing a neural network for forecasting financial and economic time series. Neurocomputing 10(3), 215-236 (1996).
8. Alexandrov, A., Monov, V.: Implementation of a service oriented architecture in smart sensor systems integration platform. In: Third Int. Conf. on Telecommunications and Remote Sensing Proceedings, vol. 1, pp. 114-120, Science and Technology Publications, Luxembourg (2014).
9. Balabanov, T.: LibreOffice Calc Three Layers Perceptron Builder, https://github.com/VelbazhdSoftwareLLC/LibreOffice-Calc-Three-Layers-Perceptron-Builder , last accessed 18 Apr 2021.
10. Tashev, T., Hristov, H.: Hierarchical interconnection modeling in computer systems. In: International Scientific Conference Communication, Electronic and Computer Systems Proceedings, pp. 215-220, Sofia (2000).
11. Bakanova, N., Bakanov, A., Atanasova, T.: Modelling human-computer interactions based on cognitive styles within collective decision-making. Advances in Science, Technology and Engineering Systems Journal 6(1), 631-635 (2021).
12. Dimitrov, W.: Software testing. Avangard, Sofia (2017).