Designing Deep Neural Network Topologies with Population-Based Metaheuristics

Grega Vrbančič, Iztok Fister Jr., Vili Podgorelec
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška 46, 2000 Maribor, Slovenia
{grega.vrbancic, iztok.fister1, vili.podgorelec}@um.si

Abstract. *Over last years the deep neural network has become one of the most popular classification methods with performance comparable and in some cases even superior to humans in the wide range of applications. However, there are still some major challenges regarding the deep neural networks. One of the biggest, with the huge impact on the classification performance, is the design of such deep neural network. In this paper, we propose a population-based metaheuristics approach for designing a deep neural network topology in a straightforward automatic manner, which performance we compare against the conventional classifiers across three different datasets. With the usage of our proposed method, unlike the conventional classifiers, we were able to achieve high classification performance with no major performance drops throughout all tested datasets.*

Keywords. machine learning, neural networks, swarm and evolutionary computation, optimization

1 Introduction

In recent years, deep neural networks (DNNs) have demonstrated performance comparable, in some cases even superior to humans in areas such as image recognition [8, 27, 6], speech recognition [7], natural language processing [5] and even in playing two-player games such as Go [21]. Such accomplishments in the application of DNNs can be largely credited to increasing computer power and a growing abundance of data. As more and more computation power becomes available, more data - bigger datasets can be used to train DNNs with more layers and more neurons at each layer, which should eventually translate to higher accuracy of such DNN models [31].

Regardless of all the major successes of utilizing the DNNs to various problems, the researchers are still facing the two major problems: the design of DNN and parameter setting for the training of the DNN. The design of DNN including the number of hidden layers, the number of neurons at each layer, the type of activation function for each layer is, generally speaking, done manually, as well as picking out the right parameters for training such DNN. However, the choice of design of DNN, as well as the choice of training parameters, has an important impact on how a DNN is going to perform a specific task. The major problem in designing the DNN topology and in picking the right training parameters is the lack of some general rules or recipes to follow, which would guarantee a good DNN performance. Basically, it depends more or less on our previous experience and trying out different DNN topologies and/or training parameters.

There are not many studies on using the population-based nature-inspired algorithms [4] for tackling the mentioned problems of using the DNNs since such approach has high computational costs. Most of the studies are focusing on solving just one part of the problem in order to keep the search space as small as possible. For example in [24, 19, 12, 28] the authors are trying to optimize the architecture of feed-forward DNNs and recurrent neural networks using evolutionary approach and in [9, 30] authors are attempting to optimize the training parameters of neural networks. Based on those encouraging results from mentioned studies, we developed a method for designing DNNs with the use of population-based metaheuristics in a straightforward automatic manner titled as GWODNN or DEDNN (Grey Wolf Optimizer for Deep Neural Network/Differential Evolution for Deep Neural Network).

The main goal of our research is to study whether a model based on DNN topology designed with our proposed population-based metaheuristics approach for designing a DNN topology in a straightforward manner, will give us generally better classification accuracy across different datasets over the conventional classifiers (e.g. k-nearest neighbor, decision tree, multilayer perceptron). The main advantages of the proposed method are the very straightforward usage and the adaptability to different datasets while achieving high classification accuracy.

The remaining of this paper is organized as follows. Section 2 briefly describes methods we used. In Section 3 we present the proposed GWODNN/DEDNN

method, whose performance of classification on various datasets are presented in Section 4. Conclusion and final remarks are gathered in Section 5.

2 Methods

2.1 Population-based metaheuristics

This subsection familiarizes readers with population-based metaheuristics, i.e. differential evolution and grey wolf optimizer.

2.1.1 Differential evolution

Differential Evolution (DE) [25] is a population-based metaheuristic algorithm that belongs to the family of evolutionary algorithms [3]. DE was introduced in 1995 by Storn and Price [25]. Because of many wins at international competitions, DE is considered as one of the most powerful algorithms appropriate for continuous optimization. DE consists of Np real-coded vectors representing the candidate solutions, as follows:

$$\mathbf{x}_i^{(t)} = (x_{i,1}^{(t)}, \ldots, x_{i,n}^{(t)}), \quad \text{for } i = 1, \ldots, Np, \tag{1}$$

where each element of the solution is in the interval $x_{i,1}^{(t)} \in [x_i^{(L)}, x_i^{(U)}]$, and $x_i^{(L)}$ and $x_i^{(U)}$ denotes the lower and upper bounds of the i-th variable, respectively. There are three different operators in DE, i.e.: mutation, crossover, and selection.

DE mutation is expressed as follows:

$$\mathbf{u}_i^{(t)} = \mathbf{x}_{r1}^{(t)} + F \cdot (\mathbf{x}_{r2}^{(t)} - \mathbf{x}_{r3}^{(t)}), \quad \text{for } i = 1, \ldots, Np, \tag{2}$$

where F denotes the scaling factor as a positive real number that scales the rate of modification while $r1,\ r2,\ r3$ are randomly selected values in the interval $1 \ldots Np$.

Crossover in DE is expressed as follows:

$$w_{i,j}^{(t+1)} = \begin{cases} u_{i,j}^{(t)} & \text{rand}_j(0,1) \leq CR \vee j = j_{rand}, \\ x_{i,j}^{(t)} & \text{otherwise}, \end{cases} \tag{3}$$

where $CR \in [0.0, 1.0]$ controls the fraction of parameters that are copied to the trial solution.

Selection is expressed mathematically as follows:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{w}_i^{(t)} & \text{if } f(\mathbf{w}_i^{(t)}) \leq f(\mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t)} & \text{otherwise}. \end{cases} \tag{4}$$

2.1.2 Grey wolf optimizer

Grey wolf optimizer or simply GWO [13] is a swarm intelligence based algorithm [4], which mimics the leadership hierarchy and hunting mechanism of grey wolves. For simulating the leadership hierarchy, four types of grey wolves are employed, i.e. alpha, beta, delta, and omega [13]. In line with this, algorithm consists of three main steps:

- searching for prey,
- encircling prey, and
- attacking prey.

Basic GWO algorithm is presented in Alg. 1. For deeper outline of GWO algorithm, readers are referred to the paper [13].

Algorithm 1 Grey wolf optimizer

```
Initialize the grey wolf population X_i (i = 1, 2, ..., n)
Initialize a, A, and C
Calculate the fitness of each search agent
X_α = the best search agent
X_β = the second best search agent
X_δ = the third best search agent
while termination_condition_not_meet do
    for each search agent do
        Update the position of the current search agent
    end for
    Update a, A, and C
    Calculate the fitness of all search agents
    Update X_α, X_β, and X_δ
end while
return X_α
```

2.2 Deep Neural Network

A standard neural network (NN) [11] consists of many simple and connected processors called neurons, each producing a sequence of real-valued activations. The input neurons get activated through sensors perceiving the environment, on the other side, other neurons get activated through weight connections from a previously active neuron. Learning of such NNs is about finding weights that make the NN exhibit the desired behavior (e.g. recognize a person from the picture). Such behavior may require long causal chains of computational stages, where each stage transforms (most often in a non-linear way) the aggregate activation of the network [20].

For decades have been around shallow NN-like models with few such stages. Models with several successive nonlinear layers of neurons date back to 50s and 60s years in a previous century. In last few years, deeper NN-like models also known as deep neural networks or DNN are gaining on the popularity.

In formal terms, a feed-forward (acyclic) and deep neural network is a tuple $N = (L, T, \Phi)$, where each of its elements is defined as follows [26]:

- $L = \{L_k | k \in \{1, ..., K\}\}$ is a set of layers, where L_1 is the input layer, L_K is the output layer, and layers in between are known as hidden layers. Each layer L_k consists of s_k nodes known as neurons.

- $T \subseteq L \times L$ is a set of connections between layers in such way that except input and output layers, each of hidden layer has an incoming connection and an outgoing connection.
- $\Phi = \{\Phi_k | \in \{2, ..., K\}\}$ is a set of activation functions, one for each non-input layer.

3 Proposed method

Our proposed method for designing DNN topology is presented in Alg. 2. The method is based on the grey wolf optimizer and differential evolution algorithm variants named GWODNN and DEDNN. GWO and DE are used to find the optimal DNN topology - in our case to find the number of hidden layers, number of neurons, dropout probability and activation function for each layer and the optimizer function for DNN. The output layer of each DNN topology is fixed, using Sigmoid activation function and Uniform function for the initialization of kernel.

Algorithm 2 Proposed method

Output: The $best$ DNN topology based on best solution

```
gwo_alg.init_bat();
while termination_condition_not_meet do
    solution = gwo_alg.get_best_solution();
    layer_num = map_batch(solution[75]);
    optimizer = map_optimizer(solution[76]);
    layers = map_layers_triples(solution[0 − (3 ∗ layer_num)]);
    fitness = train_and_eval(layers, optimizer, 100, 32);
    gwo_alg.generate_new_solution(fitness);
end while
best = create_model(gwo_alg.get_best_solution());
```

As shown in Alg. 2[1] used algorithms are producing a solution with the dimension of 77. The dimension of the problem relates to our predefined limitation with regard to the maximum number of hidden layers of DNN. In our case, the maximum number of hidden layers is set to 25 but it could be easily changed to any value. Each of those layers is defined with 3 values: number of layers, number of neurons and dropout probability. Besides definitions of layers, the last two values of a solution are defining the number of hidden layers used to create DNN topology and an optimizer function used to build the DNN model.

Therefore, the individuals in GWODNN and DEDNN are presented as real-valued vectors:

$$\mathbf{x}_i^{(t)} = (x_{i,0}^{(t)}, \ldots, x_{i,n}^{(t)}), \quad \text{for } i = 0, \ldots, Np - 1, \quad (5)$$

where each element of the solution is in the interval $x_{i,1}^{(t)} \in [0, 1]$.

[1]DEDNN is basically the same algorithm, only GWO steps are replaced by DE steps.

The real values of real-valued solution vectors are then mapped according to the equations 6, 7, 8, 9 and 10 where y_1 presents the number of hidden layers, y_2 optimization function, y_3 number of neurons in hidden layer, y_4 dropout probability of hidden layer and y_5 activation function for hidden layer. Each y_2 is member of population $O = \{sgd, rmsprop, adagrad, adadelta, adam, adamax, nadam\}$ which represents a group of available optimization functions, while y_5 is member of population $A = \{softmax, elu, selu, softplus, softsign, relu, tanh, sigmoid, hard\_sigmoid, linear\}$ which represents a group of available activation functions.

$$y_1 = \lfloor x[i] * 20 + 5 \rfloor; y_1 \in [5, 25] \quad (6)$$

$$y_2 = \begin{cases} \lfloor x[i] * 7 + 1 \rfloor; y_2 \in [1, 7] & x[i] < 1 \\ 7 & \text{otherwise,} \end{cases} \quad (7)$$

$$y_3 = \begin{cases} \lfloor x[i] * 100 + 1 \rfloor; y_3 \in [1, 100] & x[i] < 1 \\ 100 & \text{otherwise,} \end{cases} \quad (8)$$

$$y_4 = \lfloor x[i] * 30 + 20 \rfloor; y_4 \in [20, 50] \quad (9)$$

$$y_5 = \begin{cases} \lfloor x[i] * 10 + 1 \rfloor; y_5 \in [1, 10] & x[i] < 1 \\ 10 & \text{otherwise,} \end{cases} \quad (10)$$

The fitness function was defined in two variants: one using the test accuracy calculated on validation set and one using train accuracy calculated on the train set. In equations (11) and (12) are presented the formal definitions of mentioned fitness functions, where $test\_acc$ stands for accuracy calculated on validation set and $train\_acc$ stands for accuracy calculated on train set. Used implementations of GWO and DE are optimized to search for the global minimum, so our fitness functions are defined in a way, which is converting the problem of searching maximal accuracy to the problem of searching minimum as presented in formal definitions.

$$f(test\_acc) = 1 - test\_acc \quad (11)$$

$$f(train\_acc) = 1 - train\_acc \quad (12)$$

4 Experiments and Results

4.1 Datasets

For the purpose of better overall evaluation of our proposed method, we chose 3 datasets with a different number of instances, features, and distribution of classes. In the following chosen datasets are presented in more details.

4.1.1 Phishing Websites Data Set

Phishing Websites Data Set [14] from UCI Machine Learning repository [10], prepared by Mohammad et al. [15] was used for purpose of predicting phishing websites. The basic information of this dataset is presented in Table 1.

Parameter	Value
Number of features	31
Number of instances	11,055
Number of classes	2 classes
Distribution of classes	3,793 phishing websites
	7,262 legitimate websites

Table 1: The basic information about the Phishing Websites Data Set.

4.1.2 Pima Indians Diabetes Database

Pima Indians Diabetes Database [22] from Kaggle, prepared by Smith et al. [23] is dataset used for diagnostically predict whether or not a patient has diabetes. The basic information of this dataset is presented in Table 2.

Parameter	Value
Number of features	8
Number of instances	768
Number of classes	2 classes
Distribution of classes	268 with diabetes
	500 without diabetes

Table 2: The basic information about the Pima Indians Diabetes Database.

4.1.3 Bank Marketing Data Set

Bank Marketing Data Set [16] from UCI Machine Learning repository [10], prepared by Moro et al. [17] is data set related to direct marketing campaigns of a Portuguese banking institution. Dataset is used for predicting if a client will subscribe a term deposit or not. The basic information of this dataset is presented in Table 3.

Parameter	Value
Number of features	17
Number of instances	45211
Number of classes	2 classes
Distribution of classes	5289 subjects subscribed
	40922 subjects not subscribed

Table 3: The basic information about the Bank Marketing Data Set

4.2 Experimental settings

4.2.1 GWO and DE parameters

GWO and DE algorithms were initialized with parameters presented in Table 4. Algorithms were used for searching for an optimal number of hidden layers, a number of neurons, dropout probability and activation function for each hidden layer as well as for searching for optimization function used for building the model.

In our experiments, 20% of the initial dataset was used when calculating fitness using equation (12) presented in Section 3.2. When calculating fitness using equation (11) those 20% of the initial instances are further divided to train and test split in ratio 70:30 where 70% is used for training and calculating the $train\_accuracy$, while 30% is used to calculate the $test\_accuracy$ in previously mentioned equations.

Parameter	GWO	DE
Dimension of the problem	77	77
Population size	40	40
Number of function evaluations	500	500
Lower bound	0.0	0.0
Upper bound	1.0	1.0
F (Scaling factor)		0.5
CR (Crossover probability)		0.9

Table 4: Used parameter values for GWO and DE algorithms.

4.2.2 Deep Neural Network

The base of our proposed method is feed-forward NN with predefined input layer using uniform kernel initializing function and output layer with one neuron also using uniform kernel initializing function and activation function set to sigmoid. The number of the hidden layers, the configuration of each of the hid-

den layers and optimization function used for compiling the model is defined with the solution found using GWODNN and DEDNN.

4.2.3 Learning parameters

Beside the DNN topology and its layer parameters which were obtained from the solution of our GWODNN / DEDNN the other learning parameters (e.g. batch size, epochs) were picked based on our previous experience in machine learning. The training of DNN was performed with a batch size of 32 and 100 epochs. We used the same learning parameters for each of the used experimental datasets.

4.3 Results

The proposed method and experiments were implemented in Python programming language using the following frameworks and/or libraries: NiaPy [29], Keras [2] with Tensorflow [1], NumPy, Pandas and scikit-learn [18]. For all of the existing classification algorithms the implementations from scikit-learn with their default settings were used.

In order to objectively evaluate the performance of our proposed method, we followed an established methodology. All of the used datasets were initially divided into two subsets in a ratio 80:20. The smaller subset was used for finding the best DNN topology using the proposed metaheuristic methods. After obtaining the optimal DNN topology, only the remaining larger subset was used to perform the ten-fold cross-validation procedure. Namely, in order to keep the evaluation procedure as fair as possible, the 20% of the original instances, which have been already used to search for DNN topology, were not included in the final results. In the case of conventional classifiers, the standard ten-fold cross-validation approach was used on the original datasets.

Results presented in following sections are minimum, maximum, average and median of the ten-folds achieved by a specific method on test sets. With the labels *GWODNN/DEDNN - train* we refer to our GWODNN/DEDNN methods with fitness calculated based on train accuracy, while with the labels *GWODNN/DEDNN - test* we refer to our GWODNN/DEDNN methods with fitness calculated based on test accuracy. Performance of our proposed method variations were compared against multilayer perceptron classifier (MLP), with 5 hidden layers each with 100 neurons, k-nearest neighbor (KNN) with number of neighbors set to 5 classifier and decision tree classifier (DT) with minimum number of samples required to split an internal node set to 2. Beside the mentioned conventional classifiers, we also compared the performance of our proposed method variants against baseline DNN with an input layer, 2 hidden layers and an output layer. The first hidden layer is having 12 neurons while the second is having 8 neurons, both of them are utilizing ReLU activation function. On the output layer, the Sigmoid activation function is used. The same baseline DNN is used over all three datasets, which provides us with the solid baseline performance for further performance comparison against our GWODNN/DEDNN variants.

4.3.1 Performance on Phishing Websites Data Set

The results of the performance of the classifiers on the Phishing Websites Data Set is presented in Figure 1. The best performing classifier with the average accuracy of 96.9% is MLP, closely followed by the DEDNN - train and DT with the average accuracy of 96.6%. In general, all classifiers performed comparable, except the Base DNN and the KNN classifier which are lacking behind 2.8% and 2.4%. Looking closer at the performance of our proposed method variations, we can see that none of them is performing noticeably better than the others, with all the accuracies in the range of 0.6%.

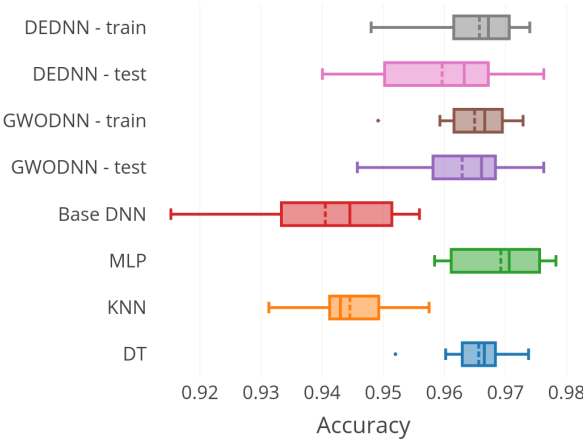

Figure 1: Comparison of accuracy between our proposed methods and other conventional classifiers using 10-fold cross validation on Phishing Websites Data Set.

4.3.2 Performance on Pima Indians Diabetes Database

The performance of classifiers on the Pima Indians Diabetes Database is presented in Figure 2. With the average accuracy of 72.6%, GWODNN - test proves to be the best performing classifier by the noticeable margin of 2.1% against the second best the KNN classifier. The worst performance achieved the DT classifier with 67.1% of average accuracy. Focusing on our methods, we can see, that the GWODNN variations did outperform the DEDNN variations by a noticeable margin of 3.9% on the case of the *test* variations and by 1.0% on the case of *train* variations. The best performing classifier on the previous dataset - MLP classifier is lagging behind the best performing classifier significantly by the margin of 3.8%.

Table 5: Average accuracy performance for all classifiers against all of the datasets

	Phishing Websites	Pima Indians Diabetes	Bank Marketing
DEDNN - train	0.966	0.692	0.895
DEDNN - test	0.960	0.687	0.889
GWODNN - train	0.965	0.702	0.889
GWODNN - test	0.963	**0.726**	**0.896**
Base DNN	0.941	0.692	0.895
MLP	**0.969**	0.688	0.889
KNN	0.945	0.705	0.882
DT	0.966	0.671	0.874

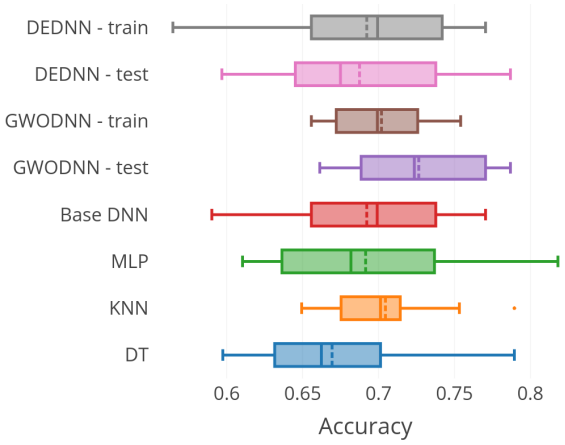

Figure 2: Comparison of accuracy between our proposed methods and other conventional classifiers using 10-fold cross validation on Pima Indians Diabetes Database

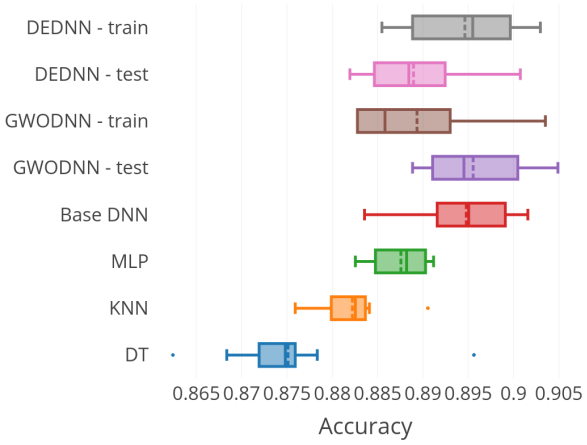

Figure 3: Comparison of accuracy between our proposed methods and other conventional classifiers using 10-fold cross validation on Bank Marketing Data Set

4.3.3 Performance on Bank Marketing Data Set

The experimental results of the performance of the classifiers on the Bank Marketing Data Set is presented in Figure 3. Looking at the boxplots, we can see that all of tested classifiers, except the DT, performed somewhat similar. The best performing classifier is GWODNN - test with the 89.6% of average accuracy. The second best classifier, leaving out the results of our method variations, is Base DNN with 89.5% lacking behind the best performing classifier by 0.1%. Comparing the performance of our method variations, we can see that results are ranging from 0.1% to 0.7%.

4.3.4 Performance comparison across all datasets

Shown in Table 5 are results of all average accuracy performances for all of the tested classifiers and on all datasets. In general, the GWODNN - test classifier performs the best in two out of three datasets, achieving the best performance on the Pima Diabetes Database and Bank Marketing Data Set, while not much (0.6%) lagging behind the MLP on the Phishing Websites Data Set. Generally speaking, all of our proposed method variants, were able to achieve high classification performance with no major performance drops throughout all tested datasets, unlike the conventional classifiers, which in some cases are lagging behind by a noticeable margin. The same performance drops throughout all tested datasets, can also be observed when comparing the performance of base DNN against out GWODNN/DEDNN variants performance.

5 Conclusions

In this paper, we presented a new approach utilizing a population-based metaheuristics algorithms to design a DNN topology in a straightforward automatic manner. The results, obtained from conducted experiments, have proven our method to be very promising, giving us high classification performance on all of the tested datasets, with no major performance drops, in comparison with conventional classifiers.

In the future, based on those encouraging results, we would like to expand our work with the use of different algorithms (e.g. Cuckoo search algorithm and Particle swarm optimization), as well as tackle the problem of time complexity using such population-based metaheuristics approach for designing a DNN topology, with parallelization methods and techniques.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2011.

[4] Iztok Fister Jr, Xin-She Yang, Iztok Fister, Janez Brest, and Dušan Fister. A brief review of nature-inspired algorithms for optimization. *Elektrotehniski vestnik*, 80(3):116–122, 2013.

[5] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[7] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Frank Hung-Fat Leung, Hak-Keung Lam, Sai-Ho Ling, and Peter Kwong-Shun Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88, 2003.

[10] M. Lichman. UCI Machine Learning Repository, 2013. Available at http://archive.ics.uci.edu/ml, Accessed: 2018-01-15.

[11] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[12] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

[13] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[14] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Phishing Websites at UCI Machine Learning Repository. Available at http://archive.ics.uci.edu/ml/datasets/Phishing+Websites, Accessed: 2018-05-17.

[15] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. An assessment of features related to phishing websites using an automated technique. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 492–497. IEEE, 2012.

[16] Sérgio Moro, Paulo Cortez, and Paulo Rita. Bank Marketing Data Set. Available at https://archive.ics.uci.edu/ml/datasets/Bank+Marketing, Accessed: 2018-05-17.

[17] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[20] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[21] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[22] Jack W Smith, JE Everhart, WC Dickson, WC Knowler, and RS Johannes. Pima Indians Diabetes Database. Available at https://www.kaggle.com/uciml/pima-indians-diabetes-database, Accessed: 2018-05-17.

[23] Jack W Smith, JE Everhart, WC Dickson, WC Knowler, and RS Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 261. American Medical Informatics Association, 1988.

[24] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[25] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[26] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *CoRR*, abs/1803.04792, 2018.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[28] Petra Vidnerová and Roman Neruda. Evolution strategies for deep neural network models design.

[29] Grega Vrbančič, Lucija Brezočnik, Uroš Mlakar, Dušan Fister, and Iztok Fister Jr. NiaPy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software*, 3, 2018.

[30] Grega Vrbančič, Iztok Fister Jr., and Vili Podgorelec. Swarm intelligence approaches for parameter setting of deep learning neural network: Case study on phishing websites classification. In *WIMS '18: Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*. ACM, accepted for publishing.

[31] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1(4):216, 2018.