Proceedings of the Federated Conference on Computer Science and Information Systems pp. 109–112 DOI: 10.15439/2017F241 ISSN 2300-5963 ACSIS, Vol. 11

Evolving Keras Architectures for Sensor Data Analysis

Petra Vidnerová
Institute of Computer Science
The Czech Academy of Sciences
Email: petra@cs.cas.cz

Roman Neruda
Institute of Computer Science
The Czech Academy of Sciences
Email: roman@cs.cas.cz

***Abstract*—Deep neural networks enjoy high interest and have become the state-of-art methods in many fields of machine learning recently. Still, there is no easy way for a choice of network architecture. However, the choice of architecture can significantly influence the network performance.**

This work is the first step towards an automatic architecture design. We propose a genetic algorithm for an optimization of a network architecture. The algorithm is inspired by and designed directly for the Keras library [1] that is one of the most common implementations of deep neural networks.

The target application is the prediction of air pollution based on sensor measurements. The proposed algorithm is evaluated on experiments on sensor data and compared to several fixed architectures and support vector regression.

I. Introduction

DEEP neural networks (DNN) architectures have become the state-of-art methods in many fields of machine learning in recent years [2], [3].

While the learning of weights of the deep neural network is done by algorithms based on the stochastic gradient descent, the choice of architecture, including a number and sizes of layers, and a type of activation function, is done manually by the user. However, the architecture has an important impact on the performance of the DNN. Some kind of expertise is needed, and usually a trial and error method is used in practice.

In this work we exploit a fully automatic design of deep neural networks. We investigate the use of genetic algorithms for evolution of a DNN architecture. There are not many studies on evolution of DNN since such approach has very high computational requirements. To keep the search space as small as possible, we simplify our model focusing on implementation of DNN in the Keras library [1] that is a widely used tool for practical applications of DNNs.

As a target application, we use a real dataset from the area of sensor networks for air pollution monitoring. We work with data from De Vito et al [4], [5].

The paper is organized as follows. Section II brings an overview of related work. Section III briefly describes the main ideas of our approach. In Section IV our algorithm GAKeras is described. Section V summarizes the results of our experiments. Finally, Section VI brings conclusion.

II. Related Work

There were quite many attempts on architecture optimization via evolutionary process (e.g. [6], [7]) in previous decades. Successful evolutionary techniques evolving the structure of feed-forward and recurrent neural networks include NEAT [8], HyperNEAT [9] and CoSyNE [10] algorithms.

On the other hand, studies dealing with evolution of deep neural networks and convolutional networks started to emerge only very recently. They usually focus only on parts of network design, due to limited computational resources. The training of one DNN usually requires hours or days of computing time, quite often utilizing GPU processors for speedup. Naturally, the evolutionary techniques requiring thousands of training trials were not considered a feasible choice. Nevertheless, there are several approaches to reduce the overall complexity of neuroevolution for DNN and provide useful and scalable algorithms.

For example, in [11] CMA-ES is used to optimize hyperparameters of DNNs. In [12] the unsupervised convolutional networks for vision-based reinforcement learning are studied, the structure of CNN is held fixed and only a small recurrent controller is evolved. However, the recent paper [13] presents a simple distributed evolutionary strategy that is used to train relatively large recurrent network with competitive results on reinforcement learning tasks.

In [14] automated method for optimizing deep learning architectures through evolution is proposed, extending existing neuroevolution methods. Authors of [15] sketch a genetic approach for evolving a deep autoencoder network enhancing the sparsity of the synapses by means of special operators. Finally, the paper [16] presents two version of an evolutionary and co-evolutionary algorithm for design of DNN with various transfer functions.

III. Our Approach

The main idea of our approach is to keep the search space as small as possible. Therefore only architecture is a subject to evolution, the weights are learnt by gradient based technique.

Further, the architecture specification is simplified. It directly follows the implementation of DNN in Keras library, where networks are defined layer by layer, each layer fully connected with the next layer. A layer is specified by number of neurons, type of an activation function (all neurons in one layer have the same type of an activation function), and type of regularization (such as dropout).

IV. Genetic Algorithm for Keras Architectures

Genetic algorithms (GA) [17], [18] represent a robust optimization technique. They work with the population of feasible solutions represented by *individuals*. Each individual is associated with *fitness* value that evaluates its quality. New generations are created iteratively by means of GA operators *selection*, *crossover* and *mutation*.

Individuals are coding feed-forward neural networks implemented as Keras model *Sequential*. The model implemented as *Sequential* is built layer by layer, similarly an individual consists of blocks representing individual layers.

$$I = ([size_1, drop_i, act_1]_1, \ldots, [size_H, drop_H, act_H]_H),$$

where H is the number of hidden layers, $size_i$ is the number of neurons in corresponding layer that is dense (fully connected) layer, $drop_i$ is the dropout rate (zero value represents no dropout), and $act_i \in \{\texttt{relu}, \texttt{tanh}, \texttt{sigmoid}, \texttt{hardsigmoid}, \texttt{linear}\}$ stands for activation function.

The operator *crossover* combines two parent individuals and produces two offspring individuals. It is implemented as one-point crossover, where the cross-point is on a border of block.

The operator *mutation* brings random changes to the individual. Each time an individual is mutated, one of the following mutation operators is randomly chosen:

- mutateLayer - introduces random changes to one randomly selected layer. One of the following operation is randomly chosen: changeLayerSize (the number of neurons is changed; either one neuron is added, one neuron is deleted, or completely new layer size is generated), changeDropOut (the dropout rate is changed), changeActivation (the activation function is changed), changeAll (the whole block is discarded and new one is randomly initialized).
- addLayer - one randomly generated block is inserted at random position.
- delLayer - one randomly selected block is deleted.

Fitness function should reflect the quality of the network represented by an individual. To assess the generalization ability of the network represented by an individual we use a crossvalidation error. The lower the crossvalidation error, the higher the fitness of the individual. Classical k-fold crossvalidation is used and the mean squared error is used as an error function.

The tournament selection is used, i.e. each turn of the tournament k individuals are selected at random and the one with the highest fitness, in our case the one with the lowest crossvalidation error, is selected.

Our implementation of the proposed GAKeras algorithm is available at [19].

V. Experiments

A. Data Set

The dataset used for our experiments consists of real-world data from the application area of sensor networks for air pollution monitoring. The data contain measurements of gas multi-sensor MOX array devices recording concentrations of several gas pollutants. There are altogether 5 sensors as inputs and 5 target output values representing concentrations of CO, NO_2, NOx, $C6H6$, and $NMHC$.

In the first experiment, the whole time period is divided into five intervals. Then, only one interval is used for training, the rest is utilized for testing. We considered five different choices of the training part selection. This task may be quite difficult, since the prediction is performed also in different parts of the year than the learning.

In the second experiments, the data are shuffled randomly and one third is used for testing and the rest for training.

Table I brings overview of data sets sizes. All tasks have 8 input values (five sensors, temperature, absolute and relative humidity) and 1 output (predicted value). All values are normalized between $\langle 0, 1\rangle$.

TABLE I
Overview of data sets sizes.

	First experiment		**Second experiment**	
Task	train set	test set	train set	test set
CO	1469	5875	4896	2448
NO2	1479	5914	4929	2464
NOx	1480	5916	4931	2465
C6H6	1799	7192	5994	2997
NMHC	178	709	592	295

B. Parameter Setup

The GAKeras algorithm was run for 100 iterations for each data set, with the population of 30 individuals.

During fitness function evaluation the network weights are trained by RMSprop for 500 epochs. For fitness evaluation, the crossvalidation error is computed. When the best individual is obtained, the corresponding network is built and trained on the whole training set and evaluated on test set.

C. Results

The testing error values of the best individuals are listed in Table II. There are average, standard deviation, minimum and maximum errors over 10 computations. The values are compared to results obtained by support vector regression (SVR) with linear, RBF, polynomial, and sigmoid kernel function. SVR was trained using Scikit-learn library [20], hyperparameters were found by grid search and crossvalidation.

The GAKeras network achieved best results in 16 cases, it in average outperforms the SVR.

Since this task does not have much training samples, also the networks evolved are quite small. The typical evolved network had one hidden layer of about 70 neurons, dropout rate 0.3 and ReLU activation function. In case of C6H6 there were two layers, about 100 neurons together, the first linear and the second ReLU without dropout.

Table III shows comparison of testing errors of GAKeras network and several fixed architectures (for example 30-10-1 stands for 2 hidden layers of 30 and 10 neurons, one neuron

TABLE II
TEST ERRORS FOR EVOLVED GAKERAS NETWORK AND SVR WITH DIFFERENT KERNEL FUNCTIONS ON THE SECOND TASK. FOR GAKERAS NETWORK THE AVERAGE, STANDARD DEVIATION, MINIMUM AND MAXIMUM OF 10 EVALUATIONS OF LEARNING ALGORITHM IS LISTED.

Testing errors

Task	GAKeras				SVR			
	avg	std	min	max	linear	RBF	Poly.	Sigmoid
CO_part1	**0.209**	0.014	0.188	0.236	0.340	0.280	0.285	1.533
CO_part2	0.801	0.135	0.600	1.048	0.614	**0.412**	0.621	1.753
CO_part3	**0.266**	0.029	0.222	0.309	0.314	0.408	0.377	1.427
CO_part4	**0.404**	0.226	0.186	0.865	1.127	0.692	0.535	1.375
CO_part5	0.246	0.024	0.207	0.286	0.348	0.207	**0.198**	1.568
NOx_part1	2.201	0.131	1.994	2.506	**1.062**	1.447	1.202	2.537
NOx_part2	1.705	0.284	1.239	2.282	2.162	1.838	**1.387**	2.428
NOx_part3	1.238	0.163	0.982	1.533	**0.594**	0.674	0.665	2.705
NOx_part4	1.490	0.173	1.174	1.835	0.864	0.903	**0.778**	2.462
NOx_part5	**0.551**	0.052	0.456	0.642	1.632	0.730	1.446	2.761
NO2_part1	**1.697**	0.266	1.202	2.210	2.464	2.404	2.401	2.636
NO2_part2	**2.009**	0.415	1.326	2.944	2.118	2.250	2.409	2.648
NO2_part3	**0.593**	0.082	0.532	0.815	1.308	1.195	1.213	1.984
NO2_part4	**0.737**	0.023	0.706	0.776	1.978	2.565	1.912	2.531
NO2_part5	1.265	0.158	1.054	1.580	1.0773	1.047	**0.967**	2.129
C6H6_part1	**0.013**	0.005	0.006	0.024	0.300	0.511	0.219	1.398
C6H6_part2	**0.039**	0.015	0.025	0.079	0.378	0.489	0.369	1.478
C6H6_part3	**0.019**	0.011	0.009	0.041	0.520	0.663	0.538	1.317
C6H6_part4	**0.030**	0.015	0.014	0.061	0.217	0.459	0.123	1.279
C6H6_part5	**0.017**	0.015	0.004	0.051	0.215	0.297	0.188	1.526
NMHC_part1	1.719	0.168	1.412	2.000	1.718	1.666	**1.621**	3.861
NMHC_part2	**0.623**	0.164	0.446	1.047	0.934	0.978	0.839	3.651
NMHC_part3	**1.144**	0.181	0.912	1.472	1.580	1.280	1.438	2.830
NMHC_part4	**1.220**	0.206	0.994	1.563	1.720	1.565	1.917	2.715
NMHC_part5	1.222	0.126	1.055	1.447	1.238	**0.944**	1.407	2.960
	16				2	2	5	0

in output layers, ReLU activation is used and dropout 0.2). The one with most (10) best results is the GAKeras network.

The results of the second experiment are listed in Table IV. In this case the GAKeras has best results in 4 cases from 5. The training sets are bigger and also the evolved architectures contained several layers. Again the dominating activation function is ReLU.

VI. CONCLUSION

We have proposed genetic algorithm for automatic design of DNNs. The algorithm was tested in experiments on the real-life sensor data set. The solutions found by our algorithm outperform SVR and selected fixed architectures. The activation function dominating in solutions is the ReLU function. Evolved architecture depends on the task size, for tasks with small number of training points networks with only one hidden layer were evolved, for bigger tasks architectures with several hidden layers were found.

In our future work we plan to extend the algorithm to work also with convolutional networks and to include more parameters, such as other types of regularization, the type of optimization algorithm, etc. The importance of this direction is supported also by the recently conceived library [21] which combines genetic algorithm with models obtained by means of Keras and TensorFlow libraries.

ACKNOWLEDGMENT

This work was partially supported by the Czech Grant Agency grant 15-18108S and institutional support of the Institute of Computer Science RVO 67985807.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

REFERENCES

[1] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015. doi: 10.1038/nature14539. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[4] S. D. Vito, E. Massera, M. Piga, L. Martinotto, and G. D. Francia, "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario," *Sensors and Actuators B: Chemical*, vol. 129, no. 2, pp. 750 – 757, 2008. doi: 10.1016/j.snb.2007.09.060. [Online]. Available: http://dx.doi.org/10.1016/j.snb.2007.09.060

[5] S. De Vito, G. Fattoruso, M. Pardo, F. Tortorella, and G. Di Francia, "Semi-supervised learning techniques in artificial olfaction: A novel approach to classification problems and drift counteraction," *Sensors Journal, IEEE*, vol. 12, no. 11, pp. 3215–3224, Nov 2012. doi: 10.1109/JSEN.2012.2192425. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2012.2192425

[6] B. u. Islam, Z. Baharudin, M. Q. Raza, and P. Nallagownden, "Optimization of neural network architecture using genetic algorithm for load forecasting," in *2014 5th International Conference on Intelligent and Advanced Systems (ICIAS)*, June 2014. doi: 10.1109/ICIAS.2014.6869528 pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ICIAS.2014.6869528

[7] J. Arifovic and R. Genay, "Using genetic algorithms to select architecture of a feedforward artificial neural network," *Physica A: Statistical Mechanics and its Applications*, vol. 289, no. 34, pp. 574 – 594, 2001. doi: 10.1016/S0378-4371(00)00479-9. [Online]. Available: http://dx.doi.org/10.1016/S0378-4371(00)00479-9

TABLE III
TESTING ERRORS FOR EVOLVED GAKERAS NETWORK AND THREE SELECTED FIXED ARCHITECTURES.

	Testing errors							
Task	GAKeras		50-1		30-10-1		30-10-30-1	
	avg	std	avg	std	avg	std	avg	std
CO_part1	**0.209**	0.014	0.230	0.032	0.250	0.023	0.377	0.103
CO_part2	0.801	0.135	0.861	0.136	**0.744**	0.142	0.858	0.173
CO_part3	0.266	0.029	**0.261**	0.040	0.305	0.043	0.302	0.046
CO_part4	**0.404**	0.226	0.621	0.279	0.638	0.213	0.454	0.158
CO_part5	**0.246**	0.024	0.283	0.072	0.270	0.032	0.309	0.032
NOx_part1	2.201	0.131	2.158	0.203	**2.095**	0.131	2.307	0.196
NOx_part2	**1.705**	0.284	1.799	0.313	1.891	0.199	2.083	0.172
NOx_part3	1.238	0.163	1.077	0.125	1.092	0.178	**0.806**	0.185
NOx_part4	1.490	0.173	**1.303**	0.208	1.797	0.461	1.600	0.643
NOx_part5	**0.551**	0.052	0.644	0.075	0.677	0.055	0.778	0.054
NO2_part1	1.697	0.266	1.659	0.250	**1.368**	0.135	1.677	0.233
NO2_part2	2.009	0.415	1.762	0.237	**1.687**	0.202	1.827	0.264
NO2_part3	0.593	0.082	0.682	0.148	**0.576**	0.044	0.603	0.069
NO2_part4	**0.737**	0.023	1.109	0.923	0.757	0.059	0.802	0.076
NO2_part5	1.265	0.158	**0.646**	0.064	0.734	0.107	0.748	0.123
C6H6_part1	0.013	0.005	**0.012**	0.006	0.081	0.030	0.190	0.060
C6H6_part2	**0.039**	0.015	0.039	0.012	0.101	0.015	0.211	0.071
C6H6_part3	**0.019**	0.011	0.024	0.007	0.091	0.047	0.115	0.031
C6H6_part4	0.030	0.015	**0.026**	0.010	0.051	0.026	0.096	0.020
C6H6_part5	**0.017**	0.015	0.025	0.008	0.113	0.025	0.176	0.058
NMHC_part1	**1.719**	0.168	1.738	0.144	1.889	0.119	2.378	0.208
NMHC_part2	0.623	0.164	**0.553**	0.045	0.650	0.078	0.799	0.096
NMHC_part3	1.144	0.181	1.128	0.089	0.901	0.124	**0.789**	0.184
NMHC_part4	1.220	0.206	1.116	0.119	0.918	0.119	**0.751**	0.096
NMHC_part5	1.222	0.126	0.970	0.094	0.889	0.085	**0.856**	0.074
	10		6		5		4	

TABLE IV
TRAINING AND TESTING ERROR OF GAKERAS NETWORK AND SVR WITH DIFFERENT KERNEL FUNCTIONS ON THE SECOND TASK. FOR GAKERAS NETWORK THE AVERAGE, STANDARD DEVIATION, MINIMUM AND MAXIMUM OF 10 EVALUATIONS OF LEARNING ALGORITHM IS LISTED.

	Testing errors							
Task	GAKeras				SVR			
	avg	std	min	max	linear	RBF	Poly.	Sigmoid
CO	**0.120**	0.004	0.114	0.125	0.200	0.152	0.157	1.511
NOx	0.295	0.021	0.273	0.334	0.328	**0.211**	0.255	1.989
NO2	**0.267**	0.009	0.248	0.280	0.494	0.368	0.406	2.046
C6H6	**0.002**	0.001	0.000	0.005	0.218	0.110	0.194	1.325
NMHC	**0.266**	0.080	0.183	0.422	0.688	0.383	0.513	3.215

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02

[9] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009. doi: 10.1162/artl.2009.15.2.15202. [Online]. Available: http://dx.doi.org/10.1162/artl.2009.15.2.15202

[10] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *Journal of Machine Learning Research*, pp. 937–965, 2008. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab/?gomez:jmlr08

[11] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," *CoRR*, vol. abs/1604.07269, 2016. [Online]. Available: http://arxiv.org/abs/1604.07269

[12] J. Koutník, J. Schmidhuber, and F. Gomez, "Evolving deep unsupervised convolutional networks for vision-based reinforcement learning," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014. doi: 10.1145/2576768.2598358. ISBN 978-1-4503-2662-9 pp. 541–548. [Online]. Available: http://dx.doi.org/10.1145/2576768.2598358

[13] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," *ArXiv e-prints*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1703.03864

[14] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[15] O. E. David and I. Greental, "Genetic algorithms for evolving deep neural networks," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO Comp '14. New York, NY, USA: ACM, 2014. doi: 10.1145/2598394.2602287. ISBN 978-1-4503-2881-4 pp. 1451–1452. [Online]. Available: http://dx.doi.org/10.1145/2598394.2602287

[16] T. H. Maul, A. Bargiela, S.-Y. Chong, and A. S. Adamu, "Towards evolutionary deep neural networks," in *ECMS 2014 Proceedings*, F. Squazzoni, F. Baronio, C. Archetti, and M. Castellani, Eds. European Council for Modeling and Simulation, 2014. doi: 10.7148/2014-0319. [Online]. Available: http://dx.doi.org/10.7148/2014-0319

[17] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[18] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (3rd ed.)*. London, UK: Springer-Verlag, 1996. ISBN 3-540-60676-9

[19] P. Vidnerová, "GAKeras," github.com/PetraVidnerova/GAKeras, 2017.

[20] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] J. Roch, "Minos," https://github.com/guybedo/minos, 2017.