Evolutionary Optimisation of Fully Connected Artificial Neural Network Topology

Conference Paper · July 2019

CITATIONS
0

READS
7

4 authors, including:

Jordan J. Bird
Aston University
8 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Aniko Ekart
Aston University
5 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Diego R. Faria
Aston University
48 PUBLICATIONS **257** CITATIONS
SEE PROFILE

Evolutionary Optimisation of Fully Connected Artificial Neural Network Topology

Jordan J. Bird[1], Anikó Ekárt[2], Christopher D. Buckingham[3], and Diego R. Faria[4]

School of Engineering and Applied Science
Aston University, Birmingham, B4 7ET, UK
{birdj1[1], a.ekart[2],c.d.buckingham[3] d.faria[4]}@aston.ac.uk

Abstract. This paper proposes an approach to selecting the amount of layers and neurons contained within Multilayer Perceptron hidden layers through a single-objective evolutionary approach with the goal of model accuracy. At each generation, a population of Neural Network architectures are created and ranked by their accuracy. The generated solutions are combined in a breeding process to create a larger population, and at each generation the weakest solutions are removed to retain the population size inspired by a Darwinian 'survival of the fittest'. Multiple datasets are tested, and results show that architectures can be successfully improved and derived through a hyper-heuristic evolutionary approach, in less than 10% of the exhaustive search time. The evolutionary approach was further optimised through population density increase as well as gradual solution max complexity increase throughout the simulation.

Keywords: Neural Networks, Evolutionary Computation, Neuroevolution, Hyperheuristics, Computational Intelligence

1 Introduction

With the increasing growth of computational resources available to both people, private companies and governments, complex deep neural networks are growing in prominence to solve problems and provide predictions through the employment of Artificial Intelligence. Deep Neural Network Topology is a heuristic problem and therefore an optimal (best) solution is unique for individual problems, and thus tuning of topology is required. This paper provides a method to optimise deep neural networks for classification via evolutionary computation, saving a considerable amount of time and resources when compared to exhaustive search processes, autonomously with little to no human input.

The main contributions of this work are twofold:

– A hyper-heuristic evolutionary algorithm to derive an ANN solution through visualisation and analysis of the full problem space for comparison

– Algorithmic dynamism inspired by nature, through growing population density, gradual increase of possible complexities, and random chance solution generation both globally and for individual neuron layers.

2 Related Work

Famous research resulted in the *'No Free Lunch'* [1] theorem for optimisation problems. That is, for those problems that can only be completely solved in polynomial time, there is no single best algorithm to find the global best solution in non-polynomial time [2]. Optimisation algorithms must be employed that are more efficient than a random search within the problem space, and of course a complete exhaustive search of the entire problem space.

Evolution of Neural Networks through Augmenting Topologies (NEAT) is an algorithm for the genetic improvement of neural networks [3]. The algorithm proposes the evolution of network layers in a non-fully connected neural network topology, that is, neurons in layer n are not all necessarily connected to layer $n+1$. The algorithm has been particularly effective in the domain of real time problems of user input to a desired action, most notably for an evolving ANN that learns to play Super Mario in real time [4] (though became overfitted after completing the first level) and a similar study involving the evolution of an ANN that autonomously plays NERO [5].

Evolution of Recurrent Neural Networks (RNN) showed very promising results, arguing that an evolutionary algorithm is far superior to the traditional method of training within the constraints of a defined topology [6]. Improvement of Neural Networks has also been successfully experimented on with Particle Swarm Optimisation (PSO) [7] (velocity based agent swarm search of the problem space).

3 Background

3.1 Evolutionary Algorithms and Neuroevolution

An evolutionary algorithm is a population-based meta-heuristic optimisation method inspired by biological evolution [8]. That is, the optimisation of an organism due to levels of reproduction beyond that which can be supported by an environment, ie. 'survival of the fittest' [9]. An organism, in this case a solution, forms a member of a generation, and will breed with other solutions to create offspring inspired by both parents. At each generation, strengths and weaknesses of population members are considered, and the environment will cause the death of the weakest (varying on implementation).

The general outline of an evolutionary optimisation algorithm is given as the following:

1. Create a 0th (initial) population by generating a set of random solutions.
2. Begin biological simulation until a specified termination requirement is met:
   (a) Select parents and breed, creating an offspring solution (a chance of random mutation may be present).
   (b) Evaluate the fitness of each individual that has not yet been evaluated
   (c) Kill off solutions based on a chosen selection method (eg. weakness), beyond the maximum size of the population.

Neuroevolution is the application of evolutionary algorithms to Artificial Neural Networks (ANN), in which parameters and/or topology are treated as inheritable traits [10]. The implementation of an Evolutionary Algorithm treats a neural network as a member of the solution population, in which classification accuracy, or low cost (cost-based error matrices and/or training time) are treated as the fitness metric.

3.2 Fully Connected Multilayer Perceptron Topology

An Artificial Neural Network (ANN) is a system of computing inspired by the biological brain [11], in that a brain's neuron (nerve cell) takes data input to the dendrites and produces an output at nerve endings [12].

Task specificity is not programmed, and thus learning must be performed. For example, in a dataset of images of 'cat' and 'dog', a neural network will autonomously learn minute differences between the two different sets of data and consider these rules when attempting to classify an image it has not seen before [13]. Learning to process data to output(s) is derived through considering examples and finding the best fit to correctly classify those examples, or produce a best distance-based single output in terms of regression problems (ie. prediction of stock market prices).

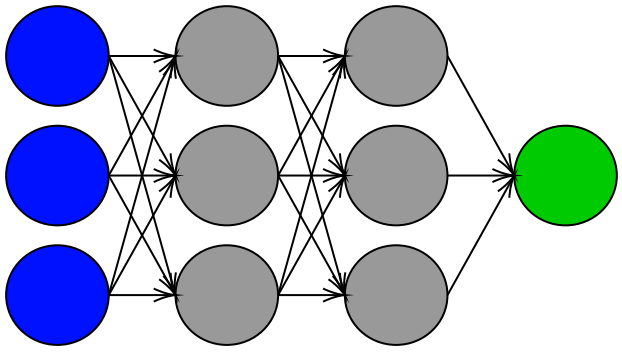

Fig. 1. A simplified diagram of a fully connected neural network. Three blue input nodes form the input layer, six grey hidden nodes form two hidden layers of three neurons, and one green node forms the regression output layer.

A fully connected topology can be seen in Fig. 1, in which all nodes of layer n are connected to all nodes of layer $n+1$. With fully connected topology assumed, this experiment is to optimise the search process of hidden layer topology, ie. the number of neurons contained within the 'grey' layers. Secondly, the experiment is also to optimise the search process of the number of hidden layers themselves (the amount of 'grey' layers in Fig. 1). Note that this is also a feedforward neural network, since all connections pass to following layers and do not form any cycles.

Backpropagation In addition to topology, the weights within an ANN are also a heuristic problem. That is, there is no 'free lunch' [1] and an optimisation algorithm must be used to fine tune weights. The training process for an Artificial Neural Network is backpropagation, a type of automatic differentiation which is a process of gradient calculation for the further derivation of weights to be used within the network [14]. Calculation of the gradient of the loss function (cost) is performed at the output compared to the real value of the data, and is passed backwards through the network starting from said output, ie. backpropagation'.

Loss is calculated dependent on the class of problem that the network is designed to solve. Absolute euclidean distance is used for a single real number output for a regression problem eg. if a house price is predicted to be £200,000 and the real value is £250,000 then the error of the output was 50,000 ie. the distance the prediction was away from the real value.
For classification, entropy is considered. Entropy is the calculation of disorder, or randomness and is given as follows:

$$E(S) = -\sum_{i=1}^{c} P_i \times log_2 P_i \tag{1}$$

For example, if a ruleset were to classify a set of 10 instances in a binary problem with an equal distribution of 5 class A and 5 class B, entropy would be 1 as the results are completely random. Lower randomness is an indication of more order within a set of rules, but without cross-validation or a separate training set, can also indicate a dataset completely memorised by an ANN that cannot be transferred to other data. This would make a prediction model worthless.

4 Method

The evolutionary process is given by Algorithms 1 and 2. Random solutions are initialised and for a set number of generations, each solution will be bred with a randomly chosen member of the population resulting in a child solution (Algorithm 2). Random chance for complete mutation of child (ie. return a random solution) and also for individual neuron layers (ie. a random number between 1 and maximum neurons) takes place, and a child is created from the two parents,

which is added to the population. After the breeding process, all untested solutions are tested, and the weakest below the maximum population are deleted. Maximum population and Maximum neuron counts are increased by the defined step, until the limit is reached.

```
Result: Array of best solutions at final generation
initialise Random solutions;
for Random solutions : rs do
    test accuracy of rs;
    set accuracy of rs;
end
while Simulating do
    for Solutions : s do
        parent2 = random Solution;
        child = breed(s, parent2); See Algorithm 2
        test accuracy of child;
        set accuracy of child;
    end
    Sort Solutions best to worst;
    for Solutions : s do
        if s index > population size then
            delete s;
        end
        increase maxPopulation by growth factor;
        increase maxNeurons by growth factor;
    end
end
Return Solutions;
```

Algorithm 1: Evolutionary Algorithm for ANN optimisation

The breeding process given in Algorithm 2 will take the length (number of layers) of one of the two parents at a 50/50 chance, and then the layers are filled in at an equal chance between the two parents, unless a parent does not have a layer at the index (ie. it is too short) in which it is taken from the parent that does. Finally a random chance occurs in which, instead of a neuron count at a layer from either parent 1 or parent 2 is returned, a random number between 1 and the maximum neuron count is returned. This process increases variability in the population by both providing a complete random solution, or a slight random solution still inspired by both parents. A final logic check is performed on whether the child happened to be identical to parent 1 or parent 2, in which case a random solution is generated.

Complexity is scaled throughout since maximum neuron and population counts increase between each generation, and thus models with very high com-

```
Result: Child solution child of parents s and parent2
random number r [1-100];
random mutation chance c;
if r ≤ c then
    child = random solution;
else
    if r ≤ 50 then
        child layerCount = s layerCount;
    else
        child layerCount = parent2 layerCount;
    end
    for Layers : l do
        random number r2 [1-100];
        if r2 ≤ 50 then
            if s layerCount ≥ l then
                child neuron at l = s neuron at l;
            else
                child neuron at l = parent2 neuron at l;
            end
        else
            if parent2 layerCount ≥ l then
                child neuron at l = parent2 neuron at l;
            else
                child neuron at l = s neuron at l;
            end
        end
        if r2 ≤ c then
            random number randomNeuron [1-maxNeurons];
            child neuron at l = randomNeuron;
        end
    end
end
if child = s OR child = parent2 then
    return random solution;
end
return child;
```

Algorithm 2: Breeding process for solution s and *parent2* derived from *Algorithm 1*

plexity and introduced later into the algorithm. This was implemented due to the computational resources required for very large networks that had low classification accuracy. The scale and maximum are manually tuned for each individual problem.

For fully-connected neural networks (solutions), 5 fold cross validation is used to create an averaged solution and thus prevent memorisation of the dataset, each ANN is allowed 500 epochs to train. The algorithm and all training/classification

was performed on an AMD FX-8320 3.5 GHz 8-Core Processor with only core applications allowed to run, the Operating System used was Windows 10. The algorithm was implemented with Java and all random numbers were generated by the Java Virtual Machine with a seed of 0.
For these experiments, hyper parameters were manually tuned to:

- A starting max population size of 4
- Population growth of 2 per generation
- Population capped at 30
- A starting max neuron count of 5 per layer
- Neuron grown of 5 per generation
- Neurons capped at 50

Two tests were performed on two different datasets. Glass Identification was a relatively simple dataset, whereas the Wine quality dataset was very complex. Due to the focus being on the search for better results, rather than the specific value of the result (high accuracy), a percentile is measured ie. where the solution exists in order of best to worst solutions. For example, if the best results were to be 2 of 100 results, they would exist within the 98th percentile. This is due to the low training times of the networks and resources available, and a benchmark of optimised search rather than the results themselves. The algorithm could then be applied to more complex neural networks (eg. 2000 epoch training time) that would be impossible to exhaustively search.

It is important to note that many problems would require far too many computational resources to viably exhaustively solve. It is for this reason that testing is performed to prove the application of the algorithm alongside a comparison of exhaustive search, with the goal that the algorithm could be further applied to problems that cannot be solved in a brute-force manner, and thus, could not be compared to a global best solution.

5 Preliminary Results

5.1 Glass Identification Dataset

A dataset of Glass Identification was acquired from the UCI Repository of Machine Learning Databases [15]. The dataset contains nine numerical attributes of the corresponding chemical compounds of the glass, along with seven classes of the glass usage (eg. 'headlamps').

The size of the dataset along with the few attributes allowed for exhaustive search to benchmark the evolutionary algorithm. Fig. 2 shows the 3D problem space for two layer neural network architectures, observation shows many local maxima for solutions. Of all solutions, the three best had identical fitness measurements and thus contributed to the top 0.12% of all solutions.

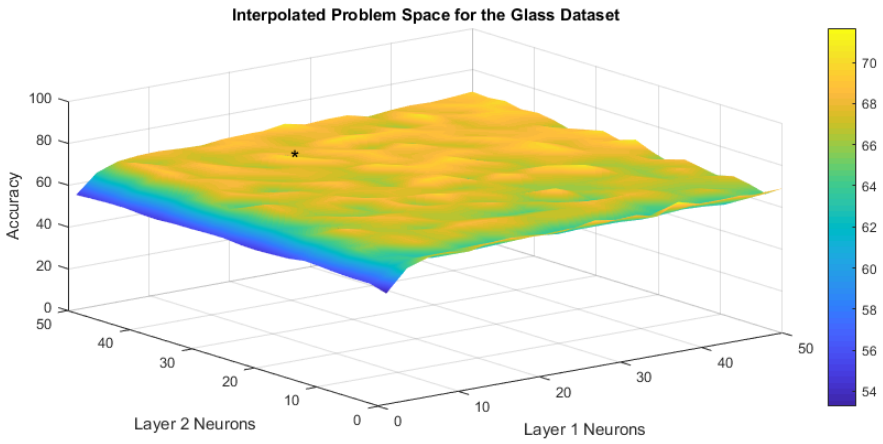

Fig. 2. 3D Interpolated Problem Space for the Glass Dataset. X and Y data are layer 1 and layer 2 neuron counts respectively. Z height shows the model accuracy, and an asterisk shows the global best solution.

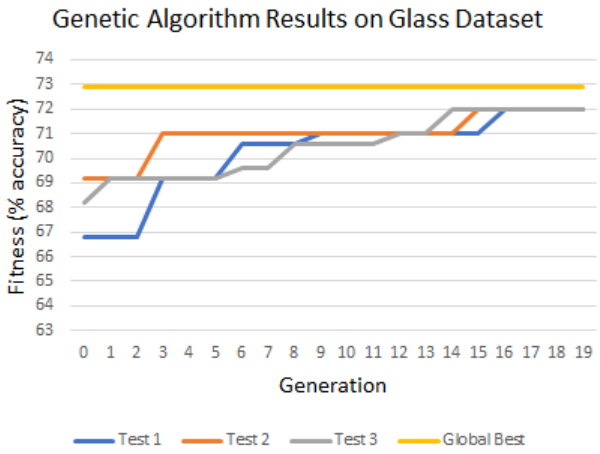

Fig. 3. Graph to show the strongest solution during three evolutionary simulations on the glass dataset.

Fig. 3 and Table I show the results of the evolutionary algorithm for the Glass Identification Dataset. Within 6.69% of the time to exhaustively search all combinations, the algorithm found the second best solution within the top 99.57 percentile of all possible results. None of the three best results were found during simulation, of which contribute 0.12% of the total number of results. Interestingly, evolutionary improvement seemed to occur at similar times during the simulations, even though the starting populi were vastly different due to random initialisation. Further work is required to study this pattern, or coincidence, of improvement.

5.2 Wine Quality Dataset

A dataset of Wine Quality was acquired from the UCI Repository of Machine Learning Databases [15]. The dataset is comprised of eleven numerical attributes which are measurements of wine contents, such as acidity and alcohol content. The data are linked to one of eleven classes, a score rating of 0-10 awarded to the wine. Fig. 4 shows the 3D problem space for two layer neural network architectures derived from an exhaustive search taking over 5 days to complete.

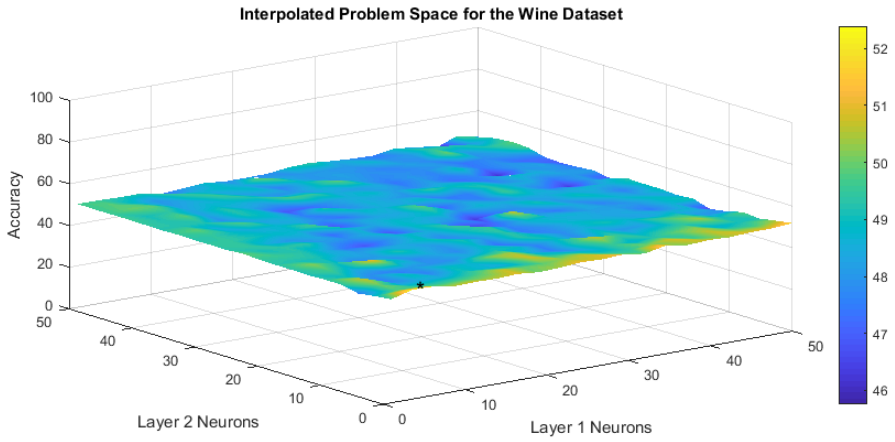

Fig. 4. 3D Interpolated Problem Space for the Wine Dataset. X and Y data are layer 1 and layer 2 neuron counts respectively. Z height shows the model accuracy, and an asterisk shows the global best solution.

5.3 Discussion

The wine dataset was far more complex than the Glass dataset, as observed by the exhaustive search time having an increase of 417,547 seconds (4 days,

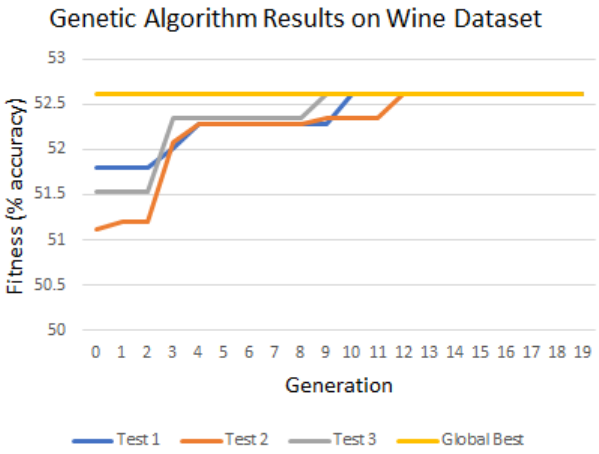

Fig. 5. Graph to show the strongest solution during three evolutionary simulations on the wine dataset.

Table 1. Results of three genetic simulations and their averages compared to exhaustive search on two separate datasets

Experiment	**Dataset**	
	Glass	***Wine***
Genetic Time 1 (S)	4352	35997
Genetic Percentile 1	99.57	99.96
Genetic Time 2 (S)	4897	36933
Genetic Percentile 2	99.57	99.96
Genetic Time 3 (S)	4675	37024
Genetic Percentile 3	99.57	99.96
Genetic Average Time (S)	4641	36651
Genetic Average Percentile	99.57	99.96
Exhaustive Time (S)	65134	482681
Exhaustive Best Percentile	99.88	99.96
Genetic vs Exhaustive Time (%)	6.69%	7.59%
Genetic vs Exhaustive Percentile	-0.31	0

19:59:07), to a total of 482681 seconds (5 days, 14:04:41) . Although this was true, a very slight percentage time increase for the wine dataset still produced very good results. This shows that the algorithm could be used for a more complex dataset, ie. a problem that cannot be realistically exhaustively searched and thus could not be verified in the same manner of the two problems in this paper.

Results of the two experiments are presented in Table I. For the glass dataset, a second best (99.57 percentile) result was found, whereas for the wine dataset, the best (99.96 percentile) was found; within 6.69% and 7.59% of the exhaustive

search time respectively. Further exploration should be performed on the glass dataset due to the late improvements at generations 13, 14, and 15, as this may possibly be due to the simulation not running for enough time.

The algorithm produced impressive results in both cases, in the case of the wine dataset saving over five days worth of computing time. The promising results presented point towards employing a genetic approach in future optimisation of artificial neural networks, due to both the time and resources saved from not performing an exhaustive search (which may be impossible). Also, the algorithm follows a logical process, presenting an autonomous system that requires little to no human intervention.

6 Next Steps

This section defines the limitations of the studies carried out and suggestions for contribution in further experiments.

This study focused on the optimisation of hidden layer neuron counts for Multilayer Perceptron Neural Networks, future work should concern the performance of optimisation in consideration of other types of Neural Networks, including the viability of this algorithm with said other Neural Network types. Additionally, only neuron and layer counts were optimised at each generation whereas parameters such as the amount of training time, momentum, and learning rate etc. should also be heuristically refined for each problem to possibly create a better solution.

The single-objective approach of this study was only concerned with the classification accuracy of a model (though more simple topology were enticed rather than their more complicated yet identical counterparts through complexity scaling), factors such as training time should be taken into account in a multi-objective optimiser to rate identical accuracy models by the computing resources they require to train, where the consumption of fewer resources would be better. This would result in a further optimised model, and would require either second comparison or a fitness score.

The evolutionary algorithm measured solution fitness by the general accuracy of the model, ie. comparing the number of correctly classified instances to the total instances in the testing dataset. For networks with an additional goal-based approach (such as medical prediction models with a focus on preventing misdiagnoses of healthy patients), a cost-based learning approach must be optimised to further achieve the model's goal. This would be implemented by the fitness value corresponding to a classification cost, and the compare operation sorting solutions lowest first rather than highest.

Due to the computational resources required for training a large volume of Multilayer Perceptrons, 5-fold cross validation was chosen for model averaging.

Related work shows that the more complex leave-one-out cross validation techniques tend to be superior in attaining higher accuracy scores [16], and therefore it would be logical to perform this experiment with those techniques. This technique was not possible with the resources available for this experiment.

Finally, the breeding algorithm is unsuitable for single hidden layer neural networks since a child produced would always be random, since it would produce a child identical to either one of its parents, therefore other breeding processes should be explored such as mid-point (average) values due to the peaks and troughs observed in figures 2 and 4 having relatively large areas and therefore smooth gradient descent.

7 Conclusion

To conclude, this study presented a high-performing genetic algorithm for the optimisation of Artificial Neural Networks which, through preliminary results, had time savings of over 90% for both a simple and a complex dataset. The algorithm has original features such as the increasing growth of the environment and thus population, as well as a cap on solution complexity that increases at each solution towards a maximum value. These were introduced to closer simulate Darwinian evolution in nature.

Future work is suggested through the further optimisation of the algorithm's hyper-parameters, as well as more complex data comparisons with their exhaustive search statistics - these experiments would be enabled with access to more computational resources.

8 Acknowledgements

This work was supported by the European Commission through the H2020 project EXCELL (https://www.excell-project.eu/), grant No. 691829.
This work was also partially supported by the EIT Health GRaCEAGE grant number 18429 awarded to C.D. Buckingham.

References

1. D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.
2. D. E. Knuth, "Postscript about np-hard problems," *ACM SIGACT News*, vol. 6, no. 2, pp. 15–16, 1974.
3. K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
4. J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, "Super mario evolution," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 156–161, IEEE, 2009.

5. K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.
6. P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.
7. Y. Shi *et al.*, "Particle swarm optimization: developments, applications and resources," in *evolutionary computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 81–86, IEEE, 2001.
8. P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016 International Conference on*, pp. 261–265, IEEE, 2016.
9. C. Darwin, *On the origin of species, 1859*. Routledge, 2004.
10. D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.
11. F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," tech. rep., CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.
12. J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.
13. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
14. Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
15. C. Blake and C. Merz, "Uci repository of machine learning databases [http://www. ics. uci. edu/ mlearn/mlrepository. html], department of information and computer science," *University of California, Irvine, CA*, vol. 55, 1998.
16. R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, pp. 1137–1145, Montreal, Canada, 1995.