Non-linear separable classification problem with MAP-Elite

Eduard Buss

University of Luebeck
eduard.buss@student.uni-luebeck.de

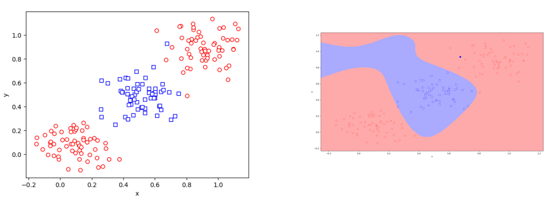
November 29, 2020

Abstract

Many of the poitns made in this book are probably wrong. It is also likely that there are considerations of critical importance that I fail to take into account, therby invalidating some or all of my conclusions - Superintelligence, Bostrom

I. Introduction

This document provides a shot summary of my first MAP-Elite approach. The overall goal is, to develop various walking patterns of a hexapod robot. To achieve this, the algorithm is first tested using a simple example where the topology as well as the weights and biases of a neural network are manipulated. A classification problem is taken which is not linear separable and therefore one hidden layer is needed (Fig. 1).



(a) (b)

Figure 1: Classificationproblem. Left: Not Classified data. Right: Classified data

Traditional search algorithms return one solution with the best performance. [Mouret and Clune, 2015] In contrast, MAP-Elite is a Quality-Diversity algorithm (QD), which not only tries to find one good solution, but many good solutions [Pugh et al., 2016]. Thus, MAP-Elite provides an archive of solutions that, in the best case, cover the entire search space.

II. Methods

All code is written in Python.

NetworkX[1] We used the framework NetworkX to implement and design a neural network (NN). NetworkX is a framework to study complex networks. Since NNs can be seen as directed graphs, this framework could be a good tool. Keep in mind that this is the beginning of a project. It may turn out that this tool complicates rather than simplifies. Nevertheless, it is possible to generate an NN with its respective neurons, connections, weights, activation functions, etc. The NN receives as input the x and y coordinates of the data. The hidden layer is given a maximum number of 4 neurons. The hidden neurons all lead to one output neuron, which returns the class affiliation. Furthermore, a bias is added to all hidden

[1] https://networkx.org/

neurons and the output neuron. All neurons of the hidden layers have a sigmoid activation function and the output neuron a simple sign function. For simplicity reasons, we will first create a fully connected parent graph of size 3x4 (layer x neurons). Then, we delete the unused neurons, two in the input layer and three in the output layer[2]. Lastly, a copy of the parent graph is made to get the working graph. The idea behind this is that as soon as a connection is added, we can just copy it from the parent graph and don't have to create a new connection, what I think is easier to implement. Additionally, the weight of a deleted connection remains in the parent graph and can be reused as soon as the connection is added again.

MAP-Elite Our approach is based on the work of Clune et al. [2013] and Mouret and Clune [2015]. The goal of this work is to find networks with high performance and different numbers of connections. Firstly, we need to define a performance measure $f(x)$ or fitness which evaluates the solution x. In our example it is simply the percentage of correctly classified data. The search space is defined by all possible values of x, which will be discussed in the mutation paragraph. It includes the weights of each connection (12-D), the bias of each neuron (7-D) and whether the neurons are connected or not (12-D) . Therefore, we have a 31-D search space. In the next step we define a n-D feature space. Following the work of Clune et al. [2013] and Mouret and Clune [2015] our space of interest consist out of the connection cost and the modularity of the NN. Now, the two dimensional feature space is discretized, so that a grid is created which can be filled with high performing solutions. The algorithm starts by producing k random individuals. We calculate directly their feature and fitness value and store them at the appropriate cell in the grid (also called archive). The respective individuals or solutions are only stored if the cell is empty or the solution it contains has a lower performance. Afterwards, we pick randomly one of the initialized solutions and modify their x-values by mutation. This can be repeated until a defined stopping criterion holds, here we use a maximum number of iterations.

[2]It is of course not necessary to first create all neurons and connections and directly delete them afterwards. But it has the advantage that you always have all neurons and connections for different scenarios

Mutations The effects of mutation are based on the work of Clune et al. [2013]. Firstly, we define a range of possible weights and biases with a defined step size. In our example we use values between -10 and 10 in steps of 0.1. Each weight and bias is mutated independently at a given probability, here we choose a probability of $2/n$ and 0.3, respectively. Where n represents the number of active connections. As soon as a modification is made, the current value will be decremented or incremented under equal probability. Another mutation operator involves adding or removing a connection. With a probability of 0.3 one of the inactive connections is added and in the same iteration one of the inactive connections is activated. The connection which will be added or removed, is chosen randomly.

feature space As mentioned before the feature space is defined by the modularity and the connection costs of the NN. Chklovskii [2004] defined an optimization problem, which minimizes the connection cost for given weights and connections. For this purpose the distances of the neurons in the respective layers are minimized (Fig. 2). Due to the minimization it can happen that neurons lie on top of each other, which is disadvantageous for the visualization. To avoid this, all neurons lying on top of each other are shifted by a constant value after the optimization. Subsequently, a second optimization is carried out in which a further constraint is introduced. This defines a minimum euclidean distance, which results in an optimized network layout without overlapping neurons (Fig. 2b). If one would add the additional constraint during the first optimization, the neurons could not move past each other, which leads to sub-optimal results (Fig. 2c).

One reason for this is that the optimization method used is sequential least squares programming, which uses a gradient to find the minimum value. The other reason is that only the x-coordinates of the neurons are changed. The y-coordinates remain constant and give information about the layer the neurons are in.

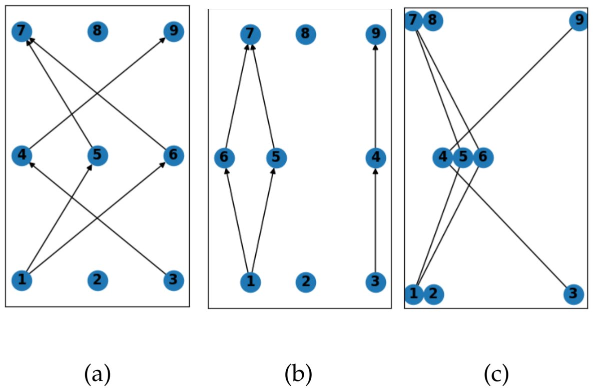


Figure 2: Minimal connection cost with constant weights. (a) Initial constellation. (b) Optimal constellation after two optimization runs. (c) Sub-optimal solution.

To identify the modularity of the given network we use Newans modularity approximation Clune et al. [2013] [Newman, 2006][Leicht and Newman, 2008]. For simplicity reasons we use an already existing implementation [3]

III. Results

We tested two different Grid resolutions. One 10x10 grid and one 50x50 grid. In both runs 200 individuals are generated randomly (Fig. 3) .

[3]https://zhiyzuo.github.io/python-modularity-maximization/

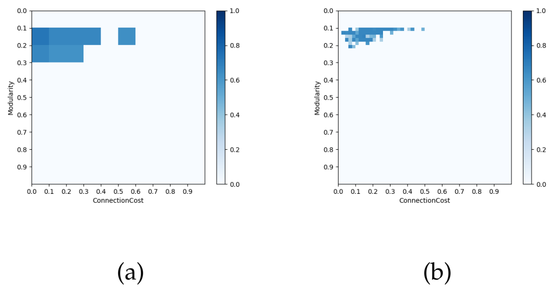
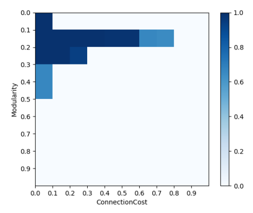


Figure 3: Initialized archive of solutions. (a) Resolution of 10x10. (b) Resoluton of 50x50

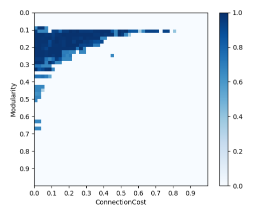
Then 20,000 iterations were performed in each experiment to manipulate the individuals and fill up the grid as best as possible (Fig. 4).

The best individual in the first run (10x10) achieves a fitness of 0.989, a modularity of 0.09 with connection costs of 0.07. Except for the connection between Neurons (1,8) and (4,8), all connections are included in the generated network with the following weights: (1, 5): -2.0, (1, 7): 7.8, (1, 6): -8.4, (4, 5): -8.6, (4, 7): 7.1, (4, 6): 7.1, (5, 10): -7.2, (6, 10): 9.9, (7, 10): -7.7 and these biases: 1: 0.0, 4: 0.0, 5: 2.8, 6: -4.4, 7: -9.5, 8: 2.3, 10: 2.6

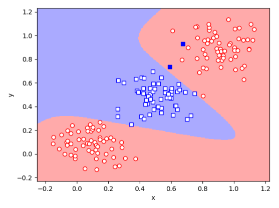
The best individual of the second run (50x50) achieves a fitness of 0.994, a modularity of 0.16 with connection costs of 0.13. Except for the connection between Neurons (1,5) and (4,6), all connections are included in the generated network with the following weights: (1, 6): -6.9, (1, 7): 2.4, (1, 8): 1.4, (4, 8): -0.7, (4, 7): 5.8, (4, 6): -0.9, (5, 10): 4.8, (6, 10): -7.1, (7, 10): -8.7, (8, 10): -4.7 and these biases: 1: 0.0, 4: 0.0, 5: -7.1, 6: 1.4, 7: -6.9, 8: -0.2, 10: -8.3.



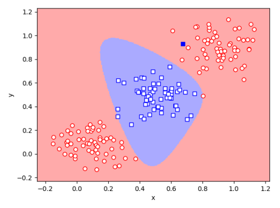
(a)



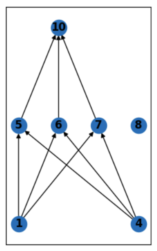
(b)



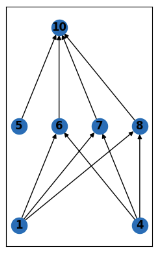
(c)



(d)



(e)



(f)

Figure 4: Results of both runs. (a) (c) (e) present results of the experiment with a lower resolution (10x10). (a) Filled archive. (c) Classification with the best solution. (e) Neural net of the best solution. (b) (d) (f)represent analog the results of the experiment with higher resolution.

IV. Discussion/Conclusion

While changing the number of connections, their weights and the bias of each neuron, we were able to find many high performance solutions. Further steps could include the determination of statistical values, for example the average fitness or the percentage of filled cells. Furthermore, the second optimization run in the calculation of the connection costs can either be discarded or further refined. Since the optimal position of the neurons depends on their weight, they can only lie directly on top of each other with the same weight. This can lead to visual overlapping of the neurons. Another interesting aspect would be the analysis of the nets in different areas of the feature map, so far, only the best of the whole archive were mentioned. The mutation operators used were taken from work that had a different example. It can be considered that other methods are more successful, as well as a higher span width or higher resolution of the weights or biases. One could also consider to implement crossover alongside the mutation and to identify its effects. Currently the upper left corner of the feature space is mostly filled. The reason for this could be the small net, with which a high modularity cannot be achieved. Possible changes, such as additional neurons, could lead to the entire feature space being covered.

References

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. pages 1–15, 2015. URL `http://arxiv.org/abs/1504.04909`.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers Robotics AI*, 3(JUL):1–17, 2016. ISSN 22969144. doi: 10.3389/frobt.2016.00040.

Jeff Clune, Jean Baptiste Mouret, and Hod Lipson. Summary of the evolutionary origins of modularity. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference Companion*, page 23, 2013. doi: 10.1145/2464576.2464596.

Dmitri B Chklovskii. Communicated by Geoffrey Goodhill Exact Solution for the Optimal Neuronal Layout Problem. Technical report, 2004.

M E J Newman. Modularity and community structure in networks. Technical report, 2006. URL `www.pnas.orgcgidoi10.1073pnas.0601602103`.

E. A. Leicht and M. E.J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11):1–5, 2008. ISSN 00319007. doi: 10.1103/PhysRevLett.100.118703.