Report for the Project "Genetic Algorithms for Image Classification via Neural Networks"

Eliane Jacobsen Gotuzzo - 20210996

Leonardo Ribeiro Teles - 20220615

Roberto Machado Velho - 20210432

Repository of the project: https://github.com/leoteles77/CIFO

May 2023

Link for the github repository of the project: https://github.com/leoteles77/CIFO

1 Introduction

This report contains the description of the project performed on applying genetic algorithms (GA) for image classification using neural networks. Contrary to the usual training mechanism of neural networks via a form of gradient descent algorithm, that updates the weights of the network along the *training process*, we decided for the use of genetic algorithms that, at each generation, chooses a set of individuals (representing the weights of the network) in a population. Such selected individuals were the ones that attained best performance, measured via a prescribed fitness function. Then, a set of genetic techniques (selection/mutation/crossover) are applied as a mechanism to generate a new population for the next generation. This new generation will again have its fitness over the data evaluated and the process repeats itself until a prescribed number of generations and the best (in terms of fitness) individual of the last generation can be taken as the set of parameters that fit the given data over the prescribed functional form of the hypothesis function (the neural network with a certain number of layers and neurons per layer in this project).

The reference data set chosen is the **MNIST** [1], a collection of images typically used for evaluating the performance of computer vision algorithms. The images represent handwritten digits from 0 to 9. The data set is composed by images of 28×28 pixels in grayscale with each pixel value ranging from 0 to 255. The images are separated in 10 different classes, representing the digits.

By using a fitness function measuring the difference between the predicted label produced by the neural network and the actual label, the classification problem can be seen as an optimization problem (a minimization one in this project), and the fitness value for each individual allow us to rank the best individuals of a generation. Since the values for the weights of the neural network are Real numbers, the optimization taking placing in this project is of the continuous form.

The goal of the project is, through the use of genetic algorithms, find the best set of weights for a neural network that can then perform the classification of the digit images. Contrary to the usual setting in machine learning, we are not exploring the capabilities of the model performing predictions over an unseen set of data (test data), the *generalization property* seen in machine learning problems. Concerning the implementations of the genetic algorithms, most of the code used in the project comes from the **Charles library** developed in the course. Some adjusts to the original code were done in order to adapt for the continuous optimization problem. New genetic operators were used, being listed in section 3. This report is organized as follows: section 2 presents a description of the problem, including details of the neural network, how individuals in a population for the genetic algorithms are repre-

[1] `http://yann.lecun.com/exdb/mnist/`

sented, and the choice of fitness functions. Section 3 lists the selection methods, mutation and crossover operators chosen for comparison. Then, section 4 describes the methodology used for comparison among the combination of methods and operators, as well as which parameters such as the probability of mutation and crossover rates were used. Finally, section 5 brings the results and conclusions for such comparisons, depicting the performance of the tested algorithms and remarks concerning the project.

2 Setting of the problem and representation of an individual

Concerning the data set, since each image contains 28×28 pixels, each one can be arranged as a vector of 784 components. This establishes the form of data is ingested in the neural network. As a form of speeding up the process of handling data before the model creation, instead of loading the images itself, a version of the data set, where the intensities of the pixels are already stored in a *csv format* file [2] is selected. The *csv* file is stored in the git repository for the project. In its original form the *train set* contains 42 thousand examples, and we select a subset of it (see section 4) for the project[3].

For the hypothesis function, a neural network, we chose one of the simplest network architectures [4]: following the input layer with 784 neurons, a hidden layer with 10 neurons, followed by the output layer with other 10 (the same number of classes for our data set) neurons. No bias was added to the neurons[5]. As for the activation functions, the hidden layer uses RELU, while the output one uses Softmax. The output layer produces a probability vector (the sum of the components add up one) of 10 components with each representing the probability of the current input data (an image represented by its pixels intensities) belonging to the respective class. The total number of weights is $784 \times 10 \ + \ 10 \times 10 \ = 7940$. Regarding the weights as an individual of the genetic algorithm, its representation is encoded as a vector of Real values with 7940 components. The initialization of individuals in the population is performed by sampling from a Standard Normal (Gaussian) distribution and applying a multiplicative scale factor of 0.3. After the initialization, the evolution along generations follows through the application of the chosen genetic techniques (see section 3). For fitness functions, cross-entropy and Brier score [1] were selected.

[2]Available in `https://www.kaggle.com/competitions/digit-recognizer/data`

[3]This subset choice was done in order to reduce the computational cost of the problem. The associated source-code is flexible enough to handle any possible size of subset

[4]Since this project concerns the exploration and comparison of efficiency concerning convergence and computational cost for different techniques of genetic algorithms, the choice of the hypothesis function as a simple neural network serves only as a toy model/architecture.

[5]Given the toy problem nature of the project.

3 Choice of genetic algorithms techniques

Genetic algorithms make use of a variety of techniques to update a population of individuals along generations. This is done via selection methods, mutation and crossover operators, and the eventual use of elitism, implemented and tested in the current project. For selection methods we chose: **tournament, rank, and fitness proportionate** - also called roulette wheel - techniques. The mutation operators picked were **inversion, scramble, and geometric semantic**. While the crossover ones were **arithmetic, blend, and geometric**.

Most of such techniques were already part of the Charles library. Geometric semantic mutation and geometric crossover were not covered in the course but are clearly defined and explored in section 3.7 of book [2] and references there in. For the scramble mutation, we refer to [3]. As for the blend crossover, we indicate [4] for the seminal paper, and [5] for an example of algorithm implementation.

4 Methodology

Given that a large search for all combinations of selection, mutation, crossover, elitism, and choice of their intrinsic parameters would be of large computational cost, and that this project serves as a pedagogical exploration of such techniques, we decided for a methodology of experiments in several steps, that, although does not explores all possibilities, still demonstrates the influence of most of possible combinations of GA techniques. Since the algorithms have a stochastic nature, we compare their performance by running the same configuration over multiple independent runs. The choice of independents runs for a given configuration along the project is 16 and they were implemented via the *joblib* python library for parallelization. The presented behavior of a given algorithm later in this project represents the average performance over the independent runs.

The experiments were conducted with a population size of 40, a crossover rate of 0.9, a mutation rate of 0.2, and a subset of the data set of size 300. Also, the default value for the number of generations was 100 and the fitness function the cross-entropy. The use or not of elitism is specified for each set of experiments. In the positive case, the number of individuals chosen for next generation, *n_elit*, is also provided. For selection technique, based on the literature on genetic algorithms, and the lectures of the course, we chose tournament. This technique tends to have a better performance than the other ones considered and implemented in the source-code, i.e., rank and fitness proportionate.

Step 1 (comparison among combinations of mutation and crossover operators):
Nine experiments (with their specifications in detail in the file *experiments.py*), ranging from Experiment 1 to 9, contemplating the combination of the three mutation and three crossover operators listed in section 3, under the parameters above, were calculated. Elitism was applied with n_elit $= 1$. Then, the two best combinations were selected for the next step.

Step 2 (comparison of selected combination of operators with variations of elitism):
At this step, we rerun the best combination of operators from the previous step, but now varying the form of elitism: not applying or applying it with n_elit $= 5$. The new experiments were numbered 10 to 13. We then compare the six algorithms.

Step 3 (use of a different fitness function - Brier score):
Given the best configuration of previous step, we compare it with its modification by just switching the choice of cost function for the Brier score.

Step 4 (final run of best configuration):
We run the selected best configuration under the default parameters but over 250 generations. Then, we inspect the distribution of values for the fitness of the best individual in the population at the last generation over the independent runs.

5 Results and conclusions

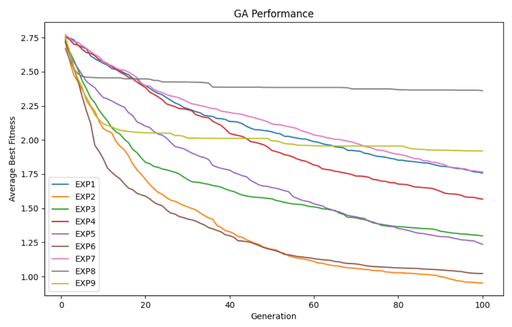
Following the methodology of previous section, we obtained the following results. For the step 1, the performances of the different combinations of mutation and crossover operators along generations are presented in Figure 1 (a). Experiment 2 (inversion mutation with arithmetic crossover) and Experiment 6 (scramble mutation with geometric crossover) showed the best results and were chosen to compose the set of experiments of step 2, where their performances with variations of elitism can be seen in Figure 1 (b). At this step, the configuration with best performance was the variation of Experiment 6 without the use of elitism.

As for step 3, the chosen best configuration of step 2, at the last generation, reached an average fitness for the best individual of 0.56, while the Brier score reached 1.31. This step of the project had more a pedagogical aspect, by showing how different fitness functions could be used and implemented in the source-code. Nevertheless, comparing such two different functions need the use of some intrinsic metric for the problem, e.g. the accuracy of the classification.

Finally, at step 4, the obtained distribution had minimal value of 0.215, while the mean was 1.10 and the standard deviation 0.88. This value for the mean, slightly higher than the one obtained at generation 100 of step 2, suggests there is a high variance for the fitness value of the best individual for a fixed value of the generation. A truly statistical analysis of it would be of great value, by increasing the number of independent runs, what was computationally unfeasible for this project.

Summarizing the results, we have that the genetic algorithm that produced the best result was the one applying scramble mutation with geometric crossover and without elitism. The impact over the convergence of each combination of operators for the genetic algorithms tested can be seen in item (a), as well as the impact of the elitism in item (b) of Figure 1.

Clearly, the results for the project could be improved. The use of a benchmark would be of

(a) 

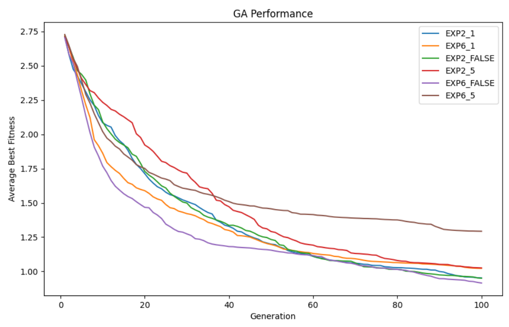
(b) 

Figure 1: (a) Average fitness evolution along generations among combinations of mutation and crossover operators. (b) Comparison of fitnesses evolution for the selected different forms of elitism.

additional value, for instance via a comparison of the same architecture of neural network trained by usual machine learning techniques. A more realistic network including bias in each neuron and under an architecture that could handle (in the sense of not suffering overfitting) the true size of the data set could be explored. Deeper statistical analysis (including statistical significance) of the performance of the genetic algorithms over the independent runs could also be done, as proposed in section 3.6 of [2].

Concerning the division of labor among the group: all members participated in the choice of problem and the selection of the classification problem using neural networks. They engaged on bibliographical research for the choice of genetic operators and techniques that could be used in this continuous optimization problem. Most of code implementation was done by L. R. Teles under live meetings with other members. Experiments were run by the different members over different computers and then collected for analyses designed by the members. Report writing was done mainly by R. M. Velho under supervision of the other members. All members accepted the final format and content of this report.

Bibliography

[1] G. W. Brier, "Verification of Forecasts Expressed in Terms of Probability," *Monthly Weather Review*, vol. 78, p. 1, Jan. 1950.

[2] L. Vanneschi and S. Silva, *Lectures on Intelligent Systems*. Springer, 2023.

[3] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[4] L. Eshelman and J. Schaffer, "Real-coded genetic algorithms and interval schemata," *Foundations of Genetic Algorithms*, vol. 2, 1993.

[5] `http://www.tomaszgwiazda.com/blendX.htm`.