Optimisation of Deep Neural Networks using a Genetic Algorithm: A Comparative Study

Tiago Gonçalves[1,2]
tiago.f.goncalves@inesctec.pt
Leonardo Capozzi[1,2]
leonardo.g.capozzi@inesctec.pt
Ana Rebelo[3]
ana.maria.s.rebelo@accenture.com
Jaime S. Cardoso[1,2]
jaime.s.cardoso@inesctec.pt

[1] Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
[2] INESC TEC
Porto, Portugal
[3] Accenture Portugal
Lisboa, Portugal

Abstract

Deep learning algorithms have been challenging human performance in several tasks. Currently, most of the methods to design the architectures of these models and to select the training hyper-parameters are still based on trial-and-error strategies. However, practitioners recognise that there is a need for tools and frameworks that can achieve high-performing models almost automatically. We addressed this challenge using a meta-heuristics approach. We implemented a genetic algorithm with a variable length chromosome with three different benchmark data sets. In this comparative study, we vary the architectures of the convolutional and the fully-connected layers and the learning rate. The best models achieve accuracy values of 98.73%, 90.81% and 54.71% on MNIST, Fashion-MNIST and CIFAR-10, respectively.

1 Introduction

The increased availability of computational power and the possibility of using graphics processing unit (GPU) cards to train and deploy deep learning algorithms in less time, allowed practitioners to implement models with complex and deep architectures and to use bigger data sets to train and validate such models. However, despite their high performances and ability to deal with raw data, these complex models require a manual selection of training parameters (commonly known as *hyper-parameters*) to converge properly. This is usually done in *trial-and-error* basis, meaning that the process may be time-consuming and tedious. To overcome this challenge, the field of automated machine learning (Auto-ML) has been created to develop tools and frameworks to achieve high-performing models almost without the need of human intervention [8]. Following the main ideas of [7], we performed a comparative study, using a variable-length chromosome genetic algorithm to optimise the hyper-parameters of deep learning models on three benchmark data sets (MNIST, Fashion-MNIST and CIFAR-10). Code is available in a public GitHub repository[1].

2 Background

2.1 Optimisation of Hyper-parameters in Deep Learning

In this work, we consider hyper-parameters all the variables that we can manually define before the beginning of the training of the algorithm (e.g., type of layers, type of activation functions, the loss function, the *optimiser*), the learning rate, the dropout rate and the number of epochs). There are several strategies of hyper-parameter optimisation, namely: *grid-search*, which consists of performing a complete search over the subset of parameters; *random-search*, which works similarly as grid-search with the difference that the search is performed randomly over the subset of parameters; *Bayesian optimisation*, which consists of building a probabilistic model based on the evaluation of the function and any available prior information, and uses that model to select the subsequent configurations of hyper-parameters to evaluate [1]; *reinforcement learning*, which consists of the utilisation of learning agents to learn the best optimisation procedures under some policy constraints [5]; and *heuristics and meta-heuristics*, which consists of the application of heuristics and/or meta-heuristics approaches.

[1]`https://github.com/TiagoFilipeSousaGoncalves/optimizing-dl-parameters-pdeec`

2.2 Genetic Algorithms

Inspired by the biological mechanisms and by Darwin's theory of evolution by natural selection, genetic algorithms rely on the fact that the fittest individuals (i.e., solutions) will prosper and will produce offspring. Also, this strategy assumes that the best characteristics (i.e., genes) are passed on the next generations. The main intuition behind this method is to ensure progress along with the generations and be sure that the best solutions will prosper through the execution of the algorithm. A standard genetic algorithm usually works as follows: 1) define a solution representation and proper encoding of the genes/chromosomes; 2) generate random solutions and evaluate them using a fitness function; 3) select the fittest individuals and apply a crossover operation to produce offspring, inducing (or not) random mutations to the next generations to increase variability.

3 Methodology

3.1 Solution Representation

Our solution representation consists of an array composed by: 1) the *convolutional layers block*, which is a tensor composed of the convolutional layers that will be part of the architecture of the model, and each convolutional layer is represented as a tensor of convolutional layer's parameters (i.e., the number of convolutional filters, the size of the convolutional kernel, the type of activation function, the dropout rate and the type of pooling function); 2) the *fully-connected layers block*, which is a tensor composed of the fully-connected layers that will be part of the architecture of the model, and each fully-connected layer is represented as a tensor of fully-connected layer's parameters (i.e., the number of output neurons, the type of activation function and the dropout rate); 3) the *learning rate* that is used to train the model.

3.2 Parameters to Optimise

Table 1 represents the type of parameters of the convolutional layers that we intended to optimise along with the values considered. Table 2 represents the type of parameters of the fully-connected layers that we intended to optimise along with the values considered. Please note that "ReLU" stands for "Rectified Linear Unit", "Tanh" stands for "hyperbolic tangent", "Max" stands for "Maximum Pooling" and "Avg" stands for "Average Pooling". Due to time constraints, we selected *adaptive moment estimation* (Adam) [2] as the only network parameter optimisation algorithm, however, with variable learning rates ($v \in \{0.001, 0.0001, 0.00001\}$).

Table 1: Names and possible values for the convolutional layers.

Name	Possible Values
Number of Convolutional Filters	$v \in \{8, 16, 32, 64, 128, 256, 512\}$
Size of the Convolutional Kernel	$v \in \{1, 3, 5, 7, 9\}$
Type of Activation Function	Identity, ReLU, Tanh
Range of Dropout Probability	$v \in [0, 1]$
Type of Pooling Function	Identity, Max, Avg

3.3 Data

To perform this comparative study we use three benchmark data sets: MNIST [4], Fashion-MNIST [6] and CIFAR-10 [3].

Table 2: Names and possible values for the fully-connected layers.

Name	Possible Values
Number of Output Neurons	$v \in [1, 100]$
Type of Activation Function	Identity, ReLU, Tanh
Range of Dropout Probability	$v \in [0, 1]$

3.4 Genetic Algorithm of Variable Chromosome Length

We start by defining f_{max} (maximum number of phases), g_{max} (number of generations), p_{max} (number of individuals of the population), i (initial chromosome length), e (number of train epochs), s (number of automatically selected individuals) and m (mutation rate). In the first phase f_0, first generation g_0, we generate $p = p_{max}$ random solutions with chromosome length $c_{len} = i$. We then train all the solutions on the training set for e epochs and save their accuracy values. After obtaining all these values, the solutions are sorted according to their accuracy values and the best s solutions are automatically selected for the next generation g_i. The solutions are then selected according to a given probability, dependent on their accuracy values (i.e., solutions with high accuracy values have high probabilities of being selected). After the selection, the solutions go through the operation of crossover. The solutions may also suffer gene mutations at a given rate of m. This processed is repeated until $g_i = g_{max}$. In the end of the last generation, we move on to the nest phase f_i and the chromosome length is updated, $c_{len} += 1$. In the phases $f_i > f_0$, the solutions are initialised with the weights of the best solution of the previous phase, through a transfer learning operation. The main idea here is to ensure that the initialisation of the solution of the next phases is not random. This way we ensure that the solutions have already some prior knowledge. One of the main reasons for this procedure is to ensure that e will not have a significant impact on the training of the solutions with longer chromosomes. Even if the solutions have different architectures, we transfer the maximum number of parameters from the reference solution. We also add an early-stopping parameter to the algorithm which stops the search if the fitness of the best model achieved so far does not increase from one phase to the next one. When the algorithm reaches convergence, we train the fittest solution for 30 epochs on the training set and test the model on the test set of each data set. To calculate the best individuals of a given generation we need to compute a fitness value for each individual. The initial fitness function that was implemented can be written as $F(s) = accuracy(s)$, where s is a solution and $accuracy(s)$ is the classification accuracy of solution s.

4 Results and Discussion

Table 3: Accuracy (%) and loss values obtained with the best model for the test set of MNIST, Fashion-MNIST and CIFAR-10.

Data set	Accuracy (%)	Loss
MNIST	98.73	0.01878
Fashion-MNIST	90.81	0.11558
CIFAR-10	54.71	1.46837

Table 3 presents the accuracy and loss values achieved in all data sets. Since the complexity of the data set increases from MNIST to CIFAR-10, the results are coherent, since the best (i.e., higher accuracy and lower loss) are achieved in the MNIST data set, followed by Fashion-MNIST and CIFAR-10. Additional results show that, in all data sets, solutions seems to lose quality from one phase to the next one (i.e., when we increase by the size of the chromosome). This becomes clearer when we observe the overall distribution of the fitnesses of the generated solutions (Figure 1 presents the distribution for MNIST database) and the distribution of the fitnesses of the generated solutions in each phase, respectively. An extended analysis of the results is publicly available in our extended report[2]. Intuitively, increasing the size of the chromosome from one phase to another (i.e., increasing the complexity of the model) would mean that the model would over-fit easier. However, we believe that this did not happen due to two possible causes: our transfer learning procedure is not benefiting the model in terms of the initialisation of its weights or the number of training epochs (e) is not enough.

[2]https://github.com/TiagoFilipeSousaGoncalves/optimizing-dl-parameters-pdeec/blob/main/report.pdf

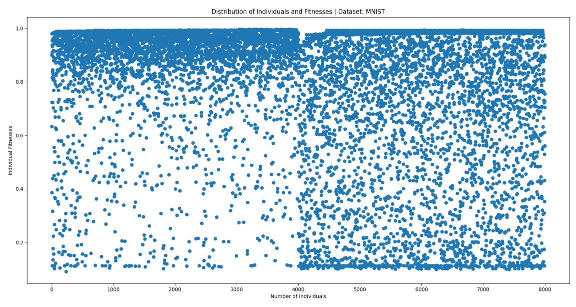

Figure 1: Overall distribution of the fitnesses of the generated solutions for MNIST database.

5 Conclusions and Future Work

Following the ideas described in [7], we implemented a genetic algorithm with variable chromosome length to achieve deep learning architectures that fit better certain data sets. The results obtained are in accordance to what has been reported in the literature and are coherent, i.e., using the same parameters to initialise the algorithm, thus, it makes sense that the best results are obtained with the simplest data set, which, in this work, is MNIST. Further work should be devoted to an ablation study in which we remove the transfer learning procedures to evaluate the impact of this operation in the quality of the generated solutions and to add a variable number of epochs and different algorithms of optimisation as hyper-parameters to achieve better training and test results. Besides, it would also be interesting to study new approaches in terms of fitness functions (e.g., include the training epochs (e)).

Acknowledgements

This work is co-financed by Component 5 - Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021 - 2026, within project HfPT, with reference 11, and by FCT – Fundação para a Ciência e a Tecnologia within the PhD grants "2020.06434.BD" and "2021.06945.BD".

References

[1] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[6] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[7] Xueli Xiao, Ming Yan, Sunitha Basodi, Chunyan Ji, and Yi Pan. Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm. *arXiv preprint arXiv:2006.12703*, 2020.

[8] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.