Global and Multi-Object Optimization Project: Combining Genetic Programming With Gradient Descent

Bruno Bonaiuto Bolivar

Course of AA 2022-2023 - DSSC

Abstract

This work describe different approaches to use gradient descent in genetic programming (GP), performing different methods. The results show when is profitable to apply GP with Gradient Descent and also when is not statistically significant.

Introduction

Genetic Algorithms are evolutionary search techniques inspired by natural selection (i.e survival of the fittest), that works with a "population" of trial solutions to a problem, frequently encoded as strings, and repeatedly select the "fitter" solutions, in this way to evolve better ones. The power of GAs is being demonstrated for an increasing range of applications. But one of the most intriguing uses of GAs - by Koza is, automatic program generation.

Genetic Programming applies Genetic algorithms to a "population" of programs - typically encoded as tree-structures. Trial programs are evaluated against a "fitness function" and the best solutions selected for modification and re evaluation. This modification-evaluation cycle is repeated until a "correct" program is produced. GP has demonstrated its potential by evolving simple programs for medical signal filters, classifying news stories, performing optical character recognition, and for target identification.

Gradient descent is optimization algorithm commonly used as black-box optimizer, in this work it will applied as a hybrid approach with GP. In order to search for improvements in the GP algorithms.

1 Genetic Programming

Genetic programming is a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done. Genetic programming is an independent method that generates a

population of computer programs to solve a problem transforming it into a new generation of programs by apply genetic operations. These genetics operations include crossover, mutation, reproduction, gene duplication, and gene deletion, processes used to transform an embryo into a developed structure. Programs are expressed in genetic programming as *syntax trees* instead of lines of code. The tree include nodes and links. The nodes indicate the instructions to execute. The links indicate the arguments for each instruction. Therefore the terminal nodes in a tree are called *functions*, and the leaves in a tree are called *terminal*. Also programs can be composed of multiples components (*subroutines*) representing the genetic program as a set of trees (one for each subroutine) grouped together under a special node called *root*. We call these (sub)trees *branches*.

1.1 Steps of Genetic Programming

Genetic programming starts from a high-level statement of the requirements of a problem in that way to produce a computer program that solves the problem, there are some well defined preparatory steps.

1. The set of terminals, for each branch of the to-be-evolved program.
2. The set of primitive functions for each branch of the to-evolved program.
3. The fitness measure of the individuals in the population.
4. Certain parameters for controlling the run.
5. The termination criterion method and the method of designating the result of the run.

The first two preparatory steps specify the ingredients available to create the computer program. The identification of the function set and terminal set for a problem is usually a easy process. For some problems, the function set might be the arithmetic functions of addition, subtraction, multiplication and division. Therefore the terminal set may be the program external inputs and numerical constants.

The third preparatory step refers to the fitness measure of the problem, it specifies what needs to be done.

The fourth and fifth preparatory steps are administrative. Implies specifying the control parameters for the run, the most important control parameter is the population size.

The fifth preparatory step consist of specifying the termination criterion and the method of designating the result to run. Including the maximum number of generations to be run.

After the user has performed the preparatory steps for a problem. The run of a genetic programming can be launched. Once the run is launched a series of steps is executed. Genetic programming typically starts with a population of randomly generated computer programs composed of the available ingredients.

Genetic programming transforms a population of computer programs into a new generation of the population by applying genetic operations. These operations are applied to individual(s) which are selected from the population. The individuals are selected to or not depending on their fitness. The iterative

transformation of the population is executed inside the main generational loop of the run of genetic programming.

At the end, the single best program in the population produced during the run (the best-so-far individual) is designated as a result of the run, if the run is successful, the result may be a solution to the problem.

2 Gradient Descent

Subsequently, Gradient descent is one of the most popular algorithms to perform optimization and also the most common way to optimize neuronal networks. Therefore Gradient descent is a long term established search/learning technique. This technique can guarantee to find a local minima for a particular task. While the local minima is not the best solution, it often meets the request of that task. Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \epsilon \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. This means, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

2.1 Gradient descent variants

There are three variants of the gradient Descent, which depends in the amount of data we use to compute the gradient of the objective function.

2.1.1 Batch gradient descent

Computes the gradient of the cost function w.r.t. to the parameters of θ of the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{1}$$

Since we need to calculate the gradients for the whole data set to perform just *one* update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. It also guarantees to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

2.1.2 Stochastic gradient descent

In contrast performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2}$$

Batch gradient descent performs redundant computation for large datasets. SGD does away with this redundancy by performing one update at a time. Therefore it is usually much faster.

2.1.3 Mini-batch gradient descent

Takes the best of both approaches and performs an update for every mini-batch of η training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3}$$

In this way it reduce the variance of the parameters updates, providing more stable convergence and it also make use of highly optimized matrix optimizations.

2.2 Gradient descent optimization algorithms

Subsequently, Various extensions have been designed for the gradient descent algorithms. Some of them are discussed below:

2.2.1 Momentum

Is a method used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients. Basically, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way. The same thing happens to our parameters updates: The momentum increases for dimensions whose gradients point in the same directions and reduces updates for dimension whose gradients change directions. The result is a faster convergence and a reduced oscillation.

2.2.2 RMSprop

The idea is divides the learning rate by a exponentially decaying average of squared gradients.

2.2.3 Adam

Adaptive Moment Estimation (Adam) computes adaptive learning rates for each parameter. It includes the Momentum method and RMSprop, along with bias correction.

3 Genetic programming with gradient descent search for multiclass object classification

The goal is to apply the gradient descent search to genetic programming for multiclass object classification problems, and to investigate whether this approach can perform better than basic GP approach in terms of training efficiency and classification performance.

The basic GP approach was performed using tree-structure to represent programs, the ramped half-and-half method was used as initialization and for the

mutation operator, moreover the proportional selection mechanism and the reproduction, crossover and mutation operators were used in the learning evolutionary process.

3.1 Preparatory steps

1. Terminals, two kinds of terminals were used: *feature terminals* and *numeric parameter terminals*.

Feature terminals from the inputs from the environment, considering mean and variances of certain regions in object cutout images.

Numeric parameter terminals are floating point numbers randomly generated using a uniform distribution at the beginning of the evolution.

2. Functions, in the function set, the four standard arithmetic and a conditional operation were used. The *if* function takes three arguments, if the first argument is negative, the *if* function returns its second argument; otherwise, it returns its third argument.

$$FuncSet = \{+, -, *, /, if\} \tag{4}$$

3. Fitness functions, classification accuracy was used on the training set as the fitness function. It refers to the number of object images that are correctly classified by the genetic program.

4. Parameters and termination criteria, the parameter values used are the following:

Parameter Names	Shape	coin1	coin2	Parameter Names	Shape	coin1	coin2
population-size	300	300	500	reproduction-rate	20%	20%	20%
initial-max-depth	3	3	3	cross-rate	50%	50%	50%
max-depth	5	6	8	mutation-rate	30%	30%	30%
max-generations	51	51	51	cross-term	15%	15%	15%
object-size	16x16	70x70	70x70	cross-func	85%	85%	85%

The learning process is terminated when one of the following conditions is met:

-The classification problem has been solved on the training set, which means, all objects of interest in the training set have been correctly classified without any missing object or false alarms for any class.

-The accuracy on the validation set starts falling down.

-The number of generations reaches the pre-defined number, *max-generations*.

3.1.1 The Data Sets

Three datasets were used to provide object classification problems in increasing difficulty.

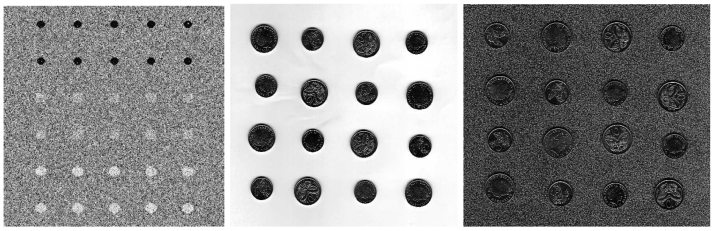

Figure 1: Example images from Shape (a), Coin1 (b) and Coin2 (c)

In order to include an easy object classification problem, a **first data set** of four different colors including a noisy background, to create classes of small objects, were cut out from the data set *shape*.

A low resolution of randomly located coins of five classes were scanned to create the **second data set** some of them were not clear to human eyes.

The **third data set** also of coins with five classes of objects with a background highly clustered, making the classification harder.

3.2 Gradient Descent Applied to Genetic Programming

The GP system will be modified by the gradient descent algorithms in this case it will just augment the existing GP system, by locally applying gradient descent search to each program in the current population in a particular generation. In other words, the normal genetic operators will not be affected or replaced.

The actual way of combining these two methods (gradient descent search and genetic programming) is by online gradient descent scheme and offline gradient descent scheme. In the Genetic Programming system, the parameters to be applied to the gradient descent are the *numeric parameter terminals* in each individual program. Assumed a continuous cost surface/function C that can be formed from a genetic program P for a given classification task based on a set of *numeric parameter terminals*.

The gradient descent search is applied to take steps "downhill" on the C from the current numeric parameter terminal T to improve the system performance.

The gradient of C is found as the vector of partial derivatives with respect to the parameters values. This gradient of vector points, in the direction of maximum-slope at the point used in the derivation. In this way changing the parameters proportionally to this vector will move the system down the surface C.

In order to do that we use O_i to represent the value of the ith numeric parameter terminal T and y to represent the output of the genetic program P, then the distance moved should be:

$$\Delta O_i = -\alpha \cdot \frac{\partial C}{\partial O_i} = -\alpha \cdot \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial O_i} \tag{5}$$

where α is a search factor. The rest of the equation are the cost function and partial derivative.

Therefore the approach applies the **online gradient descent algorithm** which in GP performs the following:

- Evaluate the program, save the outputs of all nodes in programs.
- Calculate the partial derivative of the cost functions on the program.
- Calculate the partial derivatives of the program on numeric parameter terminals by the chain rule in this way the gradients can be broken down to evaluated values and derived mathematical operators.
- Calculate the search factor α.
- Calculate the change of each numeric parameter terminal.
- Update the parameter terminals.

Subsequently, the **offline gradient descent algorithm** does a similar work but with the exception of updating the numeric parameter terminals only after a whole cycle of all examples in the training set.

3.3 Results and Conclusions

The results are delivered in this section, as previously mentioned the GP used was the basic, then the GP with online gradient descent, and lastly the GP with offline gradient descent. For all the cases, 10 runs were used and the result are the following:

Dataset	η	**Generations**			**Accuracy (%)**		
		Basic	Online	Offline	Basic	Online	Offline
Shape	0.0	9.56	-	-	99.48	-	-
	0.2	-	1.00	3.20	-	100.00	99.77
	0.4	-	1.00	2.20	-	100.00	99.86
	0.7	-	1.00	2.80	-	100.00	99.81
	1.0	-	1.00	3.30	-	100.00	99.63
	1.4	-	1.00	5.50	-	99.95	99.77

- For the first data set *Shape* there is notable result in terms of generations, in fact by just one single generation the GP with online GD was able to represent better results with 100% of accuracy for the evolutionary learning process, therefore the offline GD also achieved a better accuracy and faster training than the basic GP, but the offline GD was not as good as the online algorithm for this data set.

- Subsequently, for the *coin1* data set, the results were quite similar. Again the best performance is was performed by the online GD being better than the offline GD. Those two approaches were better than the basic GP. it was found that the method with the online gradient descent performed the best solution at 10-20 generations in all cases. On the other hand the basic method found the best program at generation 45-51 in almost all the cases. The offline algorithm

Dataset	η	Generations			Accuracy (%)		
		Basic	Online	Offline	Basic	Online	Offline
Coin1	0.0	51.00	-	-	82.12	-	-
	0.2	-	20.40	35.70	-	98.94	94.19
	0.4	-	25.40	43.40	-	98.94	93.06
	0.7	-	23.30	46.60	-	98.81	91.81
	1.0	-	36.70	47.30	-	97.69	92.44
	1.4	-	34.00	37.50	-	97.94	94.38
Coin2	0.0	51.00	-	-	73.83	-	-
	0.2	-	51.00	51.00	-	83.50	72.17
	0.4	-	51.00	51.00	-	82.83	77.67
	0.7	-	51.00	51.00	-	85.17	78.83
	1.0	-	51.00	51.00	-	86.50	84.17
	1.4	-	51.00	51.00	-	80.83	79.00

was a bit slower than the online algorithm, but still much faster than the basic GP method.

- Lastly, for the *coin2* data set, they did not improve with even more generations. The reason was because the number of features (*four*) used in the terminal set were not sufficient to perform such a difficult task.

It has been shown that for this approach that integrating gradient descent to genetic programming (GP) is better than the basic GP approach for object classification problems. The evolutionary process globally follow the beam search and locally follow the gradient descent to reach good solutions.

On all the data sets here, the two GP method with gradient descent always reached a better results than the method without GD in both classification accuracy and training generations. In particular, the best algorithm was the GP using online gradient descent search for these classification problems.

4 Genetic Programming and Gradient Descent for Binary Image Classification

The goal is to present a hybrid approach combining genetic programming (GP) and gradient descent optimisation for image classification. The performance will be compared to the baseline version (without local search) on four binary classification image dataset.

CNNs are the most used technique for image classification but there are some **limitations** as, architectures manually crafted, and models with typically low interpretability. Instead, Genetic Programming can be used for image classification and it can help to overcome the some of this limitations. GP by mimicking the natural selection and survival of the fittest to automatically search the solution for a defined problem.

Some of **the main objectives** are to look at the effects of incorporating

a local search mechanism into the evolutionary process, which is similar to the backpropagation with gradient descent in CNNs. Moreover, to discover whether the local search combined with the evolution can help to improve the solutions. Lastly, to observe the performing solutions to understand the performance.

When using the Gradient Descent as a local search in GP for classification **the function set is limited** to the four basic arithmetic operators, and also the method is not tailored for image classification, in most of the cases GP with local search outperformed significantly standard GP, with small trees overall than standard GP.

4.1 Program Structure

The program was composed by three tiers. Tier 1 (the bottom of the tree) contains the raw image as input and optional convolution and pooling operations. Tier 2 is the aggregation tier which converts the image to a numeric output, it works over a region of the image and applies measures as min, max, mean and standard deviation. Tier 3 is a classification tier, it consists of the basic operators (+, -, x, /).

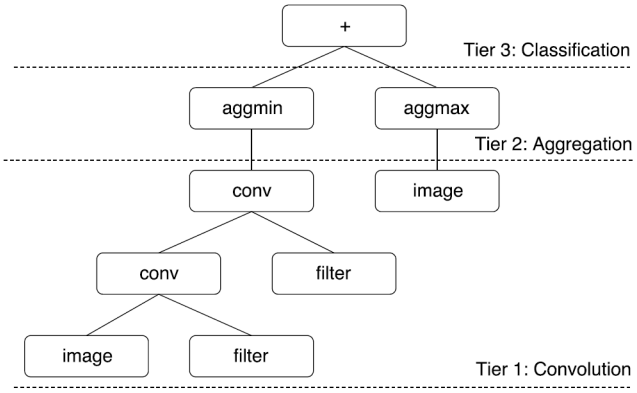

Figure 2: Example tree showing the proposed tiered architecture.

Therefore, Table 1 provide the content of the function set, input, output and a description for each function.

Since the tuning is with gradient-based optimization, the loss function and each function in the tree are differentiable, representing this as a differentiable optimization problem. The loss function measure how well are performing, similar to the fitness function for GP.

For the local search integration the gradient descent was used, specifically Stochastic Gradient Descent with mini-batches, but even with mini-batches the process of gradient was relatively slow. This means that classification accuracy is not used for local search.

Table 1: Function set

Function	Input	Output	Description
+ - x /	double	double	Performs the corresponding arithmetic operator
AggMin (·) AggMax (·) AggMean (·) AggStd (·)	(image, shape, size, position)	double	Applies the statistical measures over region of the image
convolve	(image, filter)	image	Applies the filter over the input image, and then applies ReLU activation
pool	image	image	Applies 2 x 2 max pooling to the input image

To make a fair comparison, the same training: test splits were used for each of the method. Three separated seeds were used to shuffle the data, and 30 evolutionary seeds were used for each shuffle, for a total of 90 runs. Both the local search methods and the base method had the same number of generations. However, the local search methods run for a longer time due to the inclusion of the gradient descent.

The Table 2, show the parameters and in italic the relevant for the methods with local search.

Table 2: Parameters

Parameter	Value	Parameter	Value	Parameter	Value
Population Size	1024	Crossover Rate	75 %	*Epochs*	10
Generations	50	Mutation Rate	20 %	*Number of best*	25
Tree Size	2 - 10	Reproduction Rate	5%	*Learning Rate*	0.5
Tournament Size	7			*Batch Size*	10% of training data

4.2 The Data Sets

Four widely used image dataset were used for comparison.

- Hands. The hand posture dataset.
- JAFFE. The Japanese female facial expression dataset.
- Office. The office dataset with several classes.
- CMU faces. The happy and sad faces dataset.

4.3 Results and Conclusions

The Table 3 summarize the results obtained form the datasets trialled, all the datasets used had an equal distribution of the images for each class, And its important to note that classification accuracy was not used for the local search. On the four datasets, there is not statistically significant difference. In fact, the base convGP was able to evolve good kernel alone.

This means that the inclusion of local search did not improve in terms of accuracy, the reasons were because it didn't run for long enough, also the kernel values are in a local optima so local search cannot see improvement, and potentially overfitting the training data or may be an inappropriate learning rate.

Table 3: Results of the various methods on four datasets.

		Accuracy (%)		Time	
DB	Method	Training	Testing	Training (m)	Testing (ms)
Hands	ConvGP	100.0 ± 0.00	95.88 ± 5.32	4.39 ± 4.11	85.13 ± 110.24
	ConvGP + LS	100.0 ± 0.00	96.39 ± 5.77	5.40 ± 5.38	77.01 ± 93.68
	ConvGP + LSE	100.0 ± 0.00	94.80 ± 7.00	76.38 ± 95.19	108.97 ± 140.59
JAFFE	ConvGP	99.03 ± 1.96	80.27 ± 8.29	40.92 ± 47.34	142.12 ± 187.70
	ConvGP + LS	98.24 ± 6.50	81.45 ± 8.04	79.24 ± 111.47	145.70 ± 216.87
	ConvGP + LSE	97.38 ± 7.51	81.77 ± 9.36	159.55 ± 110.61	121.07 ± 187.81
Office	ConvGP	96.20 ± 3.43	69.90 ± 9.46	28.55 ± 22.55	66.61 ± 69.17
	ConvGP + LS	92.86 ± 10.1	67.19 ± 10.6	55.92 ± 26.84	38.77 ± 40.70
	ConvGP + LSE	94.77 ± 9.04	67.90 ± 10.4	138.02 ± 187.10	66.92 ± 50.31
Faces	ConvGP	69.02 ± 4.20	44.55 ± 2.91	185.04 ± 194.76	393.41 ± 484.49
	ConvGP + LS	68.02 ± 4.68	44.40 ± 3.28	290.15 ± 286.55	259.41 ± 356.09
	ConvGP + LSE	67.81 ± 3.57	45.37 ± 3.12	226.61 ± 134.79	165.64 ± 205.36

As it is shown, the difference were not statistically significant of the two methods in terms accuracy, however another benefits were obtained during the local search as reduced tree size on average,high interpretability, architecture automatically evolved instead of manual crafted , this means that with more exploitative local search, could be likely to reach an improvement in the accuracy.

5 Conclusions

The goal of this work was to introduce to GP and Gradient Descent providing some approaches applied, in order to investigate whether hybrid approach is better than the basic GP approach.

On the section 3, the approach show that there is a significant improvement in the results, better results were obtained by applying GP with Gradient Descent search for both accuracy and training generations. Achieving the goal of the section 3.

However in section 4, the approach show that the results were not statistically significant providing no relevant difference in the results, but achieving the

goal of reducing the size of the tree, with better and automatically architecture of solutions.

This work show that GP with Gradient Descent is possible and practical depending on the application and taking into a account the limitations of local search, overfitting and learning rate.

References

[1] J. Koza and R. Poli, *Genetic Programming*, pp. 127–164. 01 2005.

[2] B. P. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Genetic programming and gradient descent: A memetic approach to binary image classification," *CoRR*, vol. abs/1909.13030, 2019.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[4] M. Zhang and W. Smart, "Genetic programming with gradient descent search for multiclass object classification," vol. 3003, pp. 399–408, 01 2004.

[5] W. B. Langdon, *Genetic Programming — Computers Using "Natural Selection" to Generate Programs*, pp. 9–42. Boston, MA: Springer US, 1998.