BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

GPM: Genetic Programming with Memory for Symbolic Regression

Bc. Tadeáš Jůza*

Abstract
This project aims to extend genetic programming to be able to capture more random and outlier values in the input data than classical genetic programming. This is achieved by extending genetic programming with a small (associative) memory to store various data points to better approximate the original data set. The method is evaluated (a) on data sets generated by standard benchmark functions for symbolic regression but contamined with randomly modified data points, and (b) as an on-chip weight generator for CNNs capable of reducing the reads from external weight memory. It was shown for some CNN layers that the method can reduce the weight memory size 24× with a 1.3% reduction in CNN accuracy.

*xjuzat00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Genetic programming (GP) is often used for symbolic regression to find an arithmetic expression approximating a given data set with a certain error. The problem becomes more difficult when the data set contains many outliers (randomly looking data points) and one requires to reproduce them with a minimal error. An obvious solution is to memorize these data points. We propose genetic programming with memory (GPM) to address this problem. This approach builds upon the foundations laid by previous research presented in [1].

GP is employed to create the mathematical expression defined by Equation 1. This can be achieved by adding memory to the GP as shown in Fig. 1.

The proposed method is evaluated (a) on data sets generated by standard benchmark functions for symbolic regression but contamined with randomly modified data points, and (b) as an on-chip weight generator for CNNs capable of reducing the reads from external weight memory. For testing purposes, Table 1 gives the standard benchmark functions derived from [2] that were contamined by randomly generated values. In case of the CNN weight generator, data sets consist of the weights of particular layers of CNN Fig. 5 for MNIST [3] image classification.

2. Poster commentary

Cartesian genetic programming (CGP) [4] was used as the GP method in GPM, see Equation 1. Fig. 1 presents the entire approach. In addition to the typical CGP, there is an associative memory (AM) and an aggregation function (Agg) which creates a key to the memory based on the input values of the system. If a value is found in memory for a given key, that value is used as the output value of the entire system, otherwise, the value generated by the function found by the CGP is used. Concatenation, xor or merging values into different intervals can be used as Agg functions.

2.1 Matematic functions

To test the proposed method, the functions in Table 1 were contamined with different percentages of random values. GPM was then run with the setup:

- size of CGP: 20 × 2 nodes
- L-back: maximum
- generations: 5 000
- population size: 5
- number of mutations: 2
- propability of mutation in memory: 0.2
- propability of constant in CGP: 0.1

The function set used in CGP depends on the function evolved and were, like the function, taken from [2].

The difference between the correct and generated output is used as the fitness for GPM and the goal is to minimize this fitness. All experiments were repeated 30×. The results for each benchmark are plotted in Fig. 3. Different colors indicate different degree of contamination by randomly generated values. The x-axis then shows the different memory sizes where the memory size is also given as a percentage of the number of points in each dataset. One can conclude that adding more memory improves (reduces) fitness.

Fig. 2 shows two solutions for a particular benchmark function Nguyen-7. Original data points are represented by ×. The function depicted in purple was generated by CGP utilizing no external memory. The function depicted in blue was generated by GPM that utilizes small memory to store the data points depicted with green. It can be seen that GPM better approximates the points from the dataset.

2.2 GPM as a weight generator

When GPM is used to generate the weights of the neural network in Fig. 5, the goal is to reduce the memory capacity needed to store the weights in external weight memory and thus reduce the communication overhead. In this task, we will compare GPM utilizing the standard 32-bit float data type with the 8-bit fixed-point encoding which is more suitable in the hardware accelerator. The CGP setup is the same as for the function approximation except the following:

- size of CGP: 20 × 10
- Function set (32bit): $+$, $-$, $*$, %, $sin(x)$, $cos(x)$, e^x, $log(x)$, x^2, x^3, $sqrt(x)$, $tan(x)$, $tanh(x)$
- Function set (8bit): x, x, max value, min value, x>>1, x>>2, x<<1, x<<2, OR, AND, XOR, $+$, $-$, $*$

For the weight generation by GPM, the associative memory can be replaced by a cheaper common memory. The reason is that the weights are always generated in the same order. As we know when a particular weight has to be read form local memory (instead of generating it by evolved expression), we developed a simple addressing circuit shown in Fig. 4. to read from the correct address.

Fig. 6 shows the classification accuracy (y-axis) when the original weights are replaced by the weights generated by GPM utilizing a fraction [%] of the original weights (x-axis). All plots show the original network accuracy, the accuracy after converting the network weights to 8bit using quantization, and the network accuracy if all generated weights were 0.

The first (left) graph shows the results of generating weights for the first convolutional layer. The second (middle) graph shows the weight generation for the second layer and the last (right) graph shows the combination of the two approaches. We can see that it is more challenging to find a suitable solution for the first layer than for the second layer. For the second layer, the solutions that do not use memory have similar accuracy as if all generated weights were replaced by 0.

Let us consider only the expression produced by GPM generating weights for the second layer, using a memory size of 10 % (here 500 values) and 8bit operations. This expression can be encoded on 36bits. The expression together with the 500 weights stored in local memory (that represent the original 5 000 weights) can be encoded on 6 660bits. Our approach leads to a 24.02× memory reduction over 32bit weights, with a 1.3 % decrease in network accuracy.

3. Conclusions

A new approach to genetic programming that uses memory has been proposed. Several benchmark problems were used to show how this approach works.

The proposed approach was used to generate weights for a CNN layer where a memory size was reduced 24× while reducing the accuracy of the CNN by 1.3 %. Better results could have been achieved if a fitness function that represents the accuracy of the CNN was used rather than just how similar the original and generated weights are.

Acknowledgements

I would like to express my sincere appreciation to my supervisor prof. Ing. Lukáš Sekanina, Ph.D. for his help and guidance throughout this project.

References

[1] Tadeáš Jůza and Lukáš Sekanina. Gpam: Genetic programming with associative memory. In *26th European Conference on Genetic Programming (EuroGP) Held as Part of EvoStar*, volume 13986 of *LNCS*, pages 68–83. Springer Nature Switzerland AG, 2023.

[2] James McDermott, David Robert White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth A. De Jong, and Una-May O'Reilly. Genetic programming needs

better benchmarks. In Terence Soule and Jason H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 791–798. ACM, 2012.

[3] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[4] Julian F. Miller. *Cartesian genetic programming*. Springer Berlin Heidelberg, 2011.