Learning in Stages: A Layered Learning Approach for Genetic Programming

Nguyen Thi Hien
Le Quy Don University
100 Hoang Quoc Viet St
Hanoi, Vietnam
Email: nguyenthihienqn@gmail.com

Nguyen Xuan Hoai
IT R&D Center,
Hanoi University
9th Km, Nguyen Trai St
Hanoi, Vietnam
Email: nxhoai@hanu.edu.vn

***Abstract*—In this paper, we propose a new layered learning approach for Genetic Programming (GP), called GPLL. Our new GPLL is an extension of the earlier work in [8] incorporating theoretically and experimentally founded components derived from progressive sampling (PS). This new version of GPLL is tested and compared with standard GP on three real-world problems. Tuned for computational efficiency, it is able to demonstrate very substantial reductions in computational cost for relatively small (and generally non-significant) reductions in generalisation accuracy. At the other extreme, computational costs are still substantially less than for GP, while generalisation accuracies are consistently slightly better.**

I. Introduction

Genetic Programming (GP), since its introduction and development by Koza, has been seen as a potential machine learning method [6]. Its main objective is to discover, by evolutionary means, relations between input and output data in the form of a function or program. GP has been applied successfully to numerous real world problems, of which many are learning tasks [9].

In machine learning (ML) [7], generalization is seen as one of the most critical properties of any learning method. To generalize effectively, any learning machine should avoid over-fitting the training data. Starting from a low base, GP generalization has attracted increasing attention from GP researchers in recent years. This has resulted in a number of publications applying traditional ML techniques to the learning processes of GP to improve its generalization capability. One such technique is layered learning (LL); it decomposes the learning task into subtasks, and then learns each subtask in layered (staged) fashion [11], [12]. Although LL has been applied to GP learning [4], [5], it has generally been used with a different purpose than generalization: combining LL with GP in learning by accurately fitting the training set part by part (for each subtask of the learning problem) in order to find precise solutions to problems in multi-robotic agent controls [4] or learning Boolean functions [5].

In [8], we proposed combining LL and incremental sampling to improve GP learning efficiency (called GPLL). However that version of the system suffered from too many ad-hoc parameter settings in setting up the layered learning. In this paper, we solve these issues by incorporating a number of techniques from progressive sampling to GPLL, and test the new learning system on a range of more complex real-world problems (by comparison with those in [8]).

The rest of the paper is organized as follows. In the next section, related work on layered learning and progressive sampling is presented. Section III outlines our new method for training GP. The experimental settings are given in sections IV. Section V presents the experimental results with discussion. The paper concludes with section VI, where some future work is highlighted.

II. Related works

This section describes two related lines of research that provide the motivation for much of this paper: layered learning and progressive sampling.

A. Layered learning

The layered learning (LL) paradigm was first formally introduced by Stone and Veloso [12] and its main idea is to solve a learning problem in a hierarchical, bottom-up fashion. First, the problem is decomposed into subtasks – often a lower-order form of the original learning problem. The learning process is then conducted in stages (layers). At each stage, the learning machine learns to solve a subtask; once the solution for the subtask has been obtained the learning machine starts learning in the next stage (layer) to solve the task in the next level, having access to the solutions learnt in the previous stages (layers).

Early work on LL mainly focused on the learning tasks in multi agent systems for playing soccer [11], or learning to over-fit for Boolean domain [5]. In [8], we presented a preliminary study of layered learning for Genetic Programming (GPLL). The objective was to investigate efficient layered training of GP using an incrementally increased sample set. This GPLL performed no better than GP on simple learning problems (with univariate target functions), but it was more efficient on more complex problems (bivariate targets). In this work, a number of parameter selections (initial raining set size, number of learning layers) were ad-hoc. For small, toy problems such as studied in that paper, such ad-hoc parameter setting can be performed through trial runs; however this is infeasible for more complex and larger data sets. Hence the

978-1-4673-0309-5/12/$31.00 ©2012 IEEE

main objective here is to extend GPLL with parameter settings derived from the theory of progressive sampling.

B. Progressive Sampling

Progressive Sampling (PS) is a school of training techniques for dealing with large data sets. As Provost et al. [10] describe PS, an algorithm is trained and retrained with increasingly larger random samples until the accuracy of the learnt model stops improving. For a training data set of size N, the task of PS is to determine a *sampling schedule* $S = \{n_0, n_1, n_2, \ldots, n_k\}$ where n_i is size of sample to be provided to the learning algorithm in stage i; it satisfies $i < j \Rightarrow n_i < n_j$ and $n_i \leq N, \forall i, j \in \{0, 1, \ldots, k\}$.

A learning machine M is then trained with S as follow. At stage i, M is executed on a randomly sample subset S_i of the training data, of size n_i. If $i > k$, or if convergence of the learning algorithm is detected (i.e generalization accuracy does not improve with more samples), PS stops the learning process and reports the result, otherwise it enters stage $i + 1$.

Provost et al. argue for three crucial issues in the success of PS:

1) The efficiency of the sampling schedule – how to determine good values for n_i ($i = 1,\ldots, k$).
2) The efficient detection of convergence – how to find the smallest possible values of each n_k, so that training on any larger sample size would not improve the model generalization (this value is known as the optimal sample size for the problem).
3) The adaptability of the sampling schedule – how the schedule can be made more flexible based on knowledge of the convergence and run-time complexity of the learning algorithm.

Gu et al. [3] demonstrated that the choice of n_0 could affect the performance of PS. They proposed a method to estimate the statistically optimal sample size (SOSS - see Section III below), and demonstrated its effectiveness.

While our GPLL approach is motivated by PS, there are some differences. PS is a general wrapper methodology for any learning machine; it does not assume that the learner can learn incrementally, so the learning is restarted at each stage. In a potentially incremental learner such as GP, this may be wasteful of computational effort. GPLL incorporates similar methods, but inside the GP structure. Because PS requires the core learning algorithm to be run multiple times, convergence detection is performed on the learning curve. GPLL uses a single run of the GP system, so convergence detection uses the error curve (on the validation set during the training process).

III. Proposed Method

Our proposed extension of GPLL [8] has the following crucial non-evolutionary components (derived from PS):

A. Choosing the Sampling Schedule

The sampling schedule uses geometric sampling, which has been proven an efficient sampling schedule in PS for any learning algorithm of polynomial run time complexity (such as GP). Following both [8] and most work on PS, we set the sampling schedule as:

$$S_g = \{2^i.n_0 : i \in (0, 1, \ldots, \lfloor \log_2 N \rfloor)\} \bigcup \{N\}$$

Thus GPLL differs from PS in that it will always continue the learning process until the data set is exhausted.

B. Determining the Initial Sample Set

To determine the initial sample set size n_0, we use Gu et al.'s algorithm [3] for calculating both SOSS and the corresponding initial training sample in one scan of the data set. Their idea is that the initial set should be that subset which most resembles the original data set. For each data subset, this is measured by the sampling quality Q, the inverse of the Kullback-Leibler (KL) divergence J between the subset and original set. To find the SOSS for a data set D of size N, we determine n sample sizes S_i spanning the range $[1, N]$. For each S_i, a corresponding sample quality Q_i is estimated. The SOSS is computed from the points (S_i, Q_i). More details of the algorithm could be found in [3].

C. Determining The Number of Learning Layers

GPLL eventually uses the whole training data set (of size N), with an initial sample set (of size n_0), so with a geometric sampling schedule, the number of learning layers l is:

$$l = \lfloor \log_2(\frac{N}{n_0}) \rfloor + 1$$

D. Stopping Criterion for Each Layer

Each learning epoch in GPLL consists of a number of successive GP generations. As in PS, it is important for GPLL to efficiently determine when to stop learning in each layer. We propose a stopping mechanism based on the effectiveness of learning in each layer. We assume that if GP learning is over-fitted for some time, then learning is ineffective. At each layer, when the new population is created from the old, the stopping criterion is called. It checks the level of over-fitting in the new population. If the amount of over-fitting has not decreased for n successive generations (a tunable parameter), the layer is ended.

To calculate the amount of overfitting, we use the algorithm of Vanneschi et al. [13]. The main idea is to measure the individual error on both training and validation sets, and compare with the best found so far. Please see [13] for the details of the algorithm.

IV. Experimental Settings

To investigate the impact of layered learning and progressive sampling on the learning efficiency of GP, we tested GPLL and standard GP on three real-world problems taken from UCIKDD [2] and StatLib [14] data sets. These problems have been widely used as benchmarks for testing the generalization performance of GP systems.[1] Table I summarizes basic information about the test problems.

[1]In this paper, we concentrate on regression problems as they have accounted for most GP applications [9]. Note that any (binary) classification problem can be transformed into a regression problem [1].

TABLE I
THE TESTED PROBLEM DATA SETS

Data sets	Features	Size	Source
Concrete Compressive Strength (Comp)	9	1030	UCI
Pollen (Poll)	5	3848	StatLib
Chscase.census (Cens)6	7	400	StatLib

The SOSS of each problem data set was calculated as in Section III. We used 20 equidistant sample sizes to cover $[1, N]$. Each numerical attribute was discretized into 10 value intervals. The number of layers was determined based on SOSS. For each problem, we built incrementally-sized training sets using the geometric sample schedule; we also constructed an independent validation set for checking the stopping criterion as in III, and a separate test set to measure the generalization accuracy of the end-of-the-run solution. The sizes for these sets are given in table II

TABLE II
DATA SETS FOR THE TEST FUNCTIONS.

Problem	Layers [Training Set Sizes]	Number of Layers	Validation Set Size	Test Set Size
Comp	[310,515]	2	257	258
Poll	[669,1343,1924]	3	962	962
Cens	[153,200]	2	100	100

To experimentally study the effect of parameter n, we conducted GPLL with $n = 3, 6, 9$, the resulting systems being denoted GPLL_3, GPLL_6, and GPLL_9. With $n = 3$, we expected the system to be efficient, but n might be too small to permit it to fully specialize to the data; for $n = 9$, we hoped to see good fit to the data, but at the potential risk of computational cost, while $n = 6$ was expected to provide a reasonable compromise between these.

The evolutionary parameter settings for all GP systems are given in Table III. We emphasize that with one exception, GPLL uses the same algorithm and settings (even, the same initial random seed) as standard GP; the only difference is that in the former, the training set (fitness cases) increases at each of the learning layers. All runs (50 for each system on each problem) were conducted on a Compaq Presario CQ3414L computer with Intel Core i3-550 Processor (4M Cache, 3.20GHz) running a Ubuntu Linux operating system. No other jobs were running at the same time.

V. RESULTS AND DISCUSSIONS

For each run of the GP settings, we recorded the generalization error (GE, test set error) of the best individual of the run, the size of that best individual, the total running time, and the first generation where the best individual was discovered. Table V shows the GE and running time, averaged over 50 runs, for all GP settings and the three problems.

We t-tested the significance of the difference in generalization error between the GPLL systems and GP, For all tests, we used a confidence level of 0.95 ($\alpha = 0.05$). For GPLL_3 and GPLL_6, since we were uncertain about their relative performance against GP, our null and alternative hypotheses were:

TABLE III
EVOLUTIONARY PARAMETER SETTINGS FOR THE GENETIC PROGRAMMING SYSTEMS

Population Size	500
Maximum generation	150
Tournament size	3
Crossover probability	0.9
Mutation probability	0.05
Initial Max depth	6
Max depth	15
Non-terminals	+, -, *, / (protected version) , sin, cos, exp, log (protected version)
Number of runs	50
Standardized fitness	mean absolute error
Elitism	Preserve the best

	p-value of test		
	GPLL_3 2-tailed	GPLL_6 2-tailed	GPLL_9 1-tailed
Comp	**0.004**	0.384	0.474
Poll	0.740	0.519	0.283
Cens	0.183	0.735	0.074

TABLE V
p-VALUES FOR T-TEST ON GENERALIZATION ERROR

- H_0 = "the average test error of the GPLL_n and GP are the same"
- H_1 = "GPLL_n and GP have different average test errors"

and thus we used two-tailed tests. Since we expected $n = 9$ to be sufficient to guarantee that each layer fully learnt the corresponding model, and therefore that GPLL_9 should outperform GP in generalization accuracy, our null and alternative hypotheses were:

- H_0 = "the average test error of GPLL_9 is no less than that of GP"
- H_1 = "the average test error of GPLL_9 is less than that of GP"

and thus we used a one-tailed test.

The p-values of the tests are given in Table V. In all t-test tables, a probability shown as zero means that the actual value is below the number of signficant digits reported. If H_0 is accepted the p-value is printed in normal face, and otherwise in bold face.

GP was significantly better at generalising than GPLL_3 on problem Comp, and non-significantly better on the other two problems; overall (since the problems are themselves independent), it was very significantly better over the three problems ($p \approx 0.0005$). For the comparison of GPLL_6 and GP, we find that the results are mixed, and no differences are significant – we cannot distinguish the generalization performance of these two algorithms on the problems, either overall, or separately. GPLL_9 was better GP than GP on all three problems, but not significantly on any. However it was significantly better over all three problems combined (($p \approx 0.0099$) – that is, while we cannot reject the null hypothesis for any specific problem, we can reject the combined null hypothesis that GPLL_9 was no

	Test Error				Running Time			
	$GPLL_3$	$GPLL_6$	$GPLL_9$	GP	$GPLL_3$	$GPLL_6$	$GPLL_9$	GP
Comp	7.88 ± 2.09	7.12 ± 1.32	6.90 ± 1.10	6.91 ± 0.97	898.18 ± 779.89	1088.80 ± 574.54	1128.06± 436.22	1682.50 ± 476.25
Poll	1.69 ± 0.33	1.63 ± 0.35	1.63 ± 0.35	1.67 ± 0.34	848.52 ± 615.25	1124.78 ± 450.43	1295.74± 589.76	3064.76 ± 927.60
Cens	1.47 ± 0.39	1.36 ± 0.26	1.31 ± 0.17	1.38 ± 0.28	75.67 ± 119.98	111.46 ± 92.24	186.06± 135.81	418.00 ± 198.62

TABLE IV
GENERALIZATION ERROR AND RUNNING TIME OF GPLL ($n = 3, 6, 9$) AND GP ON THREE TEST PROBLEMS

better GP than GP on any of the problems.

	p-value of test		
	$GPLL_3$	$GPLL_6$	$GPLL_9$
Comp	**0.000**	**0.000**	**0.000**
Poll	**0.000**	**0.000**	**0.000**
Cens	**0.000**	**0.000**	**0.000**

TABLE VI
p-VALUES FOR T-TEST ON RUN TIME

For running time, we started from the assumption that GPLL would speed up processing, so that we used one-tailed tests throughout. Our hypotheses were:

- H_0 = "the average running time of the GPLL is no lower than that of GP"
- H_1 = "the average running time of GPLL is lower than that of GP"

In all cases, GPLL was very significantly faster than GP. For example, on average the accuracy of the solution obtained by $GPLL_3$ for problem Cens degraded about 6% compared to GP, but its training time was about 6 times faster.

VI. CONCLUSION

In this paper, we extended the GPLL of [8] to use parameter-setting techniques derived from progressive sampling (PS). With these extensions, GPLL became an efficiently tunable learning system. At one end of the range of tuning settings explored, GPLL performed similarly to standard GP in generalization, with very substantial improvements in running time. At the other extreme, GPLL was still substantially less computationally expensive, but also generated consistently (and sometimes significantly) better generalizations.

In future, we plan to test GPLL on more real-world complex problems with larger data sets. More sophisticated and efficient stopping criteria for learning layers, and improvements to the whole learning process in earlier layers, are also important objectives for our near-future work.

VII. ACKNOWLEDGMENT

We are grateful to Prof. Robert Ian McKay for his various useful discussions and comments related to this work. This work was funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2011.08.

REFERENCES

[1] U. Bhowan, M. Johnston, and Silva, "Evolving ensembles in multi-objective genetic programming for classification with unbalanced data," in *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*. Dublin, Ireland: ACM, 12-16 jul 2011, pp. 1331–1338.

[2] C. Blake, E. Keogh, and C. J. Merz, "Uci repository of machine learning databases," Department of Information and Computer Science, University of California, Irvine, CA, 1998. [Online]. Available: http://www.ics.uci.edu/ mlearn/MLRepository.html

[3] B. Gu, B. Liu, F. Hu, and H. Liu, "Efficiently determine the starting sample size for progressive sampling," in *DMKD*, 2001.

[4] S. M. Gustafson, "Layered learning in genetic programming for a co-operative robot soccer problem," Master's thesis, Kansas State University, Manhattan, KS, USA, dec 2000. [Online]. Available: http://www.cs.nott.ac.uk/ smg/research/publications/msthesis-2000.ps

[5] D. Jackson and A. P. Gibbons, "Layered learning in boolean gp problems," in *Proceedings of the 10th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, Eds., vol. 4445. Valencia, Spain: Springer, 11-13 apr 2007, pp. 148–159.

[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[7] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[8] T. H. Nguyen, X. H. Nguyen, and R. I. McKay, "A study on genetic programming with layered learning and incremental sampling," in *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, A. E. Smith, Ed., IEEE Computational Intelligence Society. New Orleans, USA: IEEE Press, 5-8 jun 2011, pp. 1184–1190.

[9] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. [Online]. Available: http://www.gp-field-guide.org.uk

[10] F. J. Provost, D. Jensen, and T. Oates, "Efficient progressive sampling," in *KDD*, 1999, pp. 23–32.

[11] P. Stone, "Layered learning in multi-agent systems," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, dec 1998. [Online]. Available: http://www.cs.cmu.edu/ pstone/thesis/

[12] P. Stone and M. M. Veloso, "Layered learning," in *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, ser. Lecture Notes in Artificial Intelligence, vol. 1810. Springer, 2000, pp. 369–381.

[13] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*. Portland, Oregon, USA: ACM, 7-11 jul 2010, pp. 877–884.

[14] P. Vlachos, "Statlib project repository," Department of Statistics, Carnegie Mellon University, Pittsburgh, PA, 2000. [Online]. Available: http://lib.stat.cmu.edu