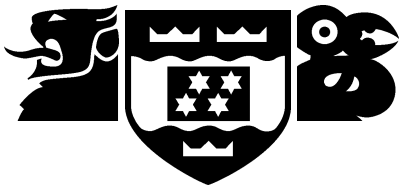
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

Evolving Weights in Genetic Programs Using Gradient Descent

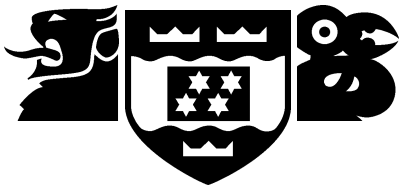
Will Smart and Mengjie Zhang

Technical Report CS-TR-04/11
August 2004

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Evolving Weights in Genetic Programs Using Gradient Descent

Will Smart and Mengjie Zhang

Technical Report CS-TR-04/11
August 2004

Abstract

This paper describes an approach to the use of gradient descent search in tree based genetic programming for object recognition problems. To learn better partial programs, a weight parameter is introduced in each link between every two nodes in a program tree, so that a change of a weight corresponds to a change of the effect of the sub-program tree. Inside a particular generation, weight changes are learnt locally by gradient descent search, but the whole evolution process is still carried out across different generations globally by the genetic beam search. This approach is examined and compared with the basic genetic programming approach without gradient descent on three object classification problems of various difficulty. The results suggest that the new approach outperforms the basic approach on all problems.

Author Information

Will Smart is a postgraduate student in computer science and Mengjie Zhang is an academic staff member in computer science. Both authors are in the School of Mathematical and Computing Sciences, Victoria University of Wellington, New Zealand.

1 Introduction

Since the early 1990s, there have been a number of reports on applying genetic programming (GP) techniques to object recognition problems [1, 3, 6, 10, 11, 12, 13]. Typically, these GP systems used either high level or low level image features as the terminal set, arithmetic and conditional operators as the function set, and classification accuracy, error rate or similar measures as the fitness function. During the evolutionary process, selection, crossover and mutation operators were applied to the genetic beam search to find good solutions. While some of these GP systems achieved reasonable even good results, others did not perform well. In addition, they usually spent a long time for training/learning good programs for a particular task. One reason for this is that they did not use the existing heuristics inside individual programs, for example, the gap between the actual outputs of the programs and the target outputs. Because of this, the evolutionary process was often stopped when a maximum number of generations was reached, rather than an ideal solution was found.

Gradient descent is a long term established search/learning technique and commonly used to train multilayer feed forward neural networks [7]. A main property of this search is that it can use heuristics in a neural network or other methods effectively. This algorithm can also guarantee to find a local minima for a particular task. While the local minima might not be the best solution, it often meets the request of that task. A main characteristic of gradient descent search is that the solutions can be improved locally but steadily.

Gradient descent search has been applied to numeric terminals [15] or constants [8] of genetic programs in GP. In these approaches, gradient descent search is applied to individual programs locally in a particular generation and the constants in a program are updated accordingly.

1.1 Goals

The goal of this research is to develop a new approach to the use of gradient descent search in genetic programming for multiclass object classification problems. Instead of searching for certain good constants or numeric terminals in a genetic program, gradient descent in this approach will used to learn partial structures in genetic programs. Specifically, we will investigate:

- how to apply gradient descent to genetic programs so that better partial (sub) programs can be learned.
- how the gradient descent search cooperate with the genetic beam search.
- whether this new approach can do a good enough job on a sequence of object classification problems of increasing difficulty.
- whether this new approach outperforms the basic GP approach.

The rest of the paper is organised as follows. Section 2 describes the GP approach to object classification, including terminals, functions, fitness function and genetic operators. Section 3 specifies the gradient descent search algorithm in our approach. Section 4 gives the image data sets used in the experiments. Section 5 presents experimental results and section 6 draws the conclusions and gives future work.

2 GP Applied to Object Classification

In the basic GP approach, we used the tree-structure to represent genetic programs [4]. The ramped half-and-half method was used for generating programs in the initial population and

for the mutation operator [2]. Three-way tournament selection mechanism and the reproduction [14], crossover and mutation operators [5] were used in the learning and evolutionary process.

2.1 Terminals

For object classification problems, terminals generally correspond to image features. In this approach, we used pixel level, domain independent statistical features (referred to as *pixel statistics*) as terminals and we expect the GP evolutionary process can automatically select features that are relevant to a particular domain to construct good genetic programs.

Four pixel statistics are used in this approach: the average intensity of the whole object cutout image, the variance of intensity of the whole object cutout image, the average intensity of the central local region, and the variance of intensity of the central local region. Since the range of these four features are quite different, we linearly normalised these feature values into the range [-1, 1] based on all object image examples to be classified.

In addition, we also used some constants as terminals. These constants are randomly generated using a uniform distribution. To be consistent with the feature terminals, we also set the range of the constants as [-1, 1].

2.2 Functions

$$FuncSet = \{+, -, \times, pdiv, if\}$$

The $+$, $-$, and $\times$ operators have their usual meanings — addition, subtraction and multiplication, while *pdiv* represents "protected" division which is the usual division operator except that a divide by zero gives a result of zero. Each of these functions takes two arguments. The *if* function takes three arguments. The first argument, which can be any expression, constitutes the condition. If the first argument is negative, the *if* function returns its second argument; otherwise, it returns its third argument.

2.3 Fitness Function

We used classification accuracy on the training set as the fitness function. The classification accuracy of a genetic program classifier refers to the number of training set object images that are correctly classified by the genetic program classifier as a proportion of the total number of object images in the training set.

In this approach, we used a variant version of the *program classification map* [14] to perform object classification. This variation situates class regions sequentially on the floating point number line. The object image will be classified to the class of the region that the program output with the object image input falls into. Class region boundaries start at some negative number, and end at the same positive number. Boundaries between the starting point and the end point are allocated with an identical interval of 1.0. For example, a five class problem would have the following classification map.

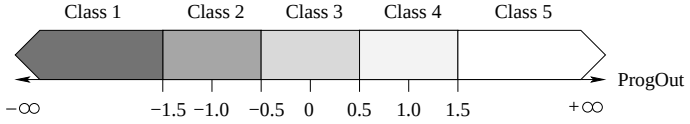


Figure 1: An example classification map.

3 Gradient Descent Applied to Weights in Program Structure

Our new GP approach used the same program representation, terminals, functions, fitness function and genetic operators as the basic GP approach as described in section 2. However, the new approach also introduced a new parameter, *weight*, to the links between every two nodes in a genetic program so that the gradient descent search can be applied in order to learn better programs locally. Due to the introduction of the weight parameter, the program trees, functions and the genetic operators need some corresponding changes. This section describes the weight parameter, corresponding changes associated with the weight parameter, and the method of using gradient descent search for updating these weights.

3.1 Genetic Programs with Weights

Unlike the standard genetic programs, this approach introduces a weight parameter between every two adjacent nodes (between every two adjacent functions or between a function and an adjacent terminal), as shown in figure 2.

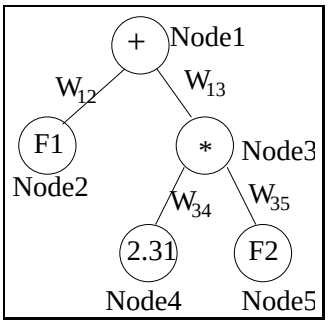


Figure 2: An example evolved program with weights.

The weights operate in a way similar to the weights in Neural Networks. Before being passed on to the higher part of the tree, the result of each subtree in the evolved program is multiplied by the value of the weight associated with the link. For example, if w_{ij} is the weight between node i and node j, then the input of node i from the branch of node j will be the weight w_{ij} times the output o_j of node j. A value of 1 for the weight corresponds to the standard structure of genetic programs in the basic GP approach.

Due to the introduction of the weights, the program structures are changed. Accordingly, the functions we used in the basic GP approach will also need corresponding changes. For example, if a node of the addition function (+) with two child nodes a_1 and a_2 and corresponding weights w_1 and w_2, the output of the addition function would be $w_1a_1+w_2a_2$, which is different from that in the basic GP approach, which is just $a_1 + a_2$. The new definitions of the functions in the function set shown in figure 3 are described in table 1.

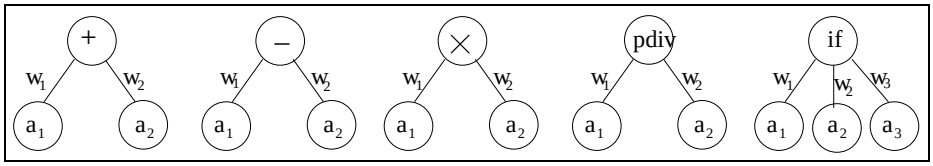


Figure 3: Functions with the new program structure.

To cope with this change, the genetic operators in this approach are also redefined in a similar way. In the new crossover operators, the two sub-programs including the nodes and the weights are swapped after two crossover points are randomly selected from the two

Table 1: New definitions of the functions in the function set.

Functions	Function output values
$+$	$w_1a_1 + w_2a_2$
$-$	$w_1a_1 - w_2a_2$
$\times$	$w_1a_1w_2a_2$
$pdiv$	$\frac{w_1a_1}{w_2a_2}$
if	$(w_1a_1 < 0)?w_2a_2 : w_3a_3$

parent programs. In mutation, after a mutation point is randomly chosen, the sub-program was replaced by a new sub-program with randomly mutated nodes and corresponding weights. Examples for the new crossover and mutation are shown in figure 4. Not that the reproduction operator still copy the selected number of best programs from the current generation into the next generation.

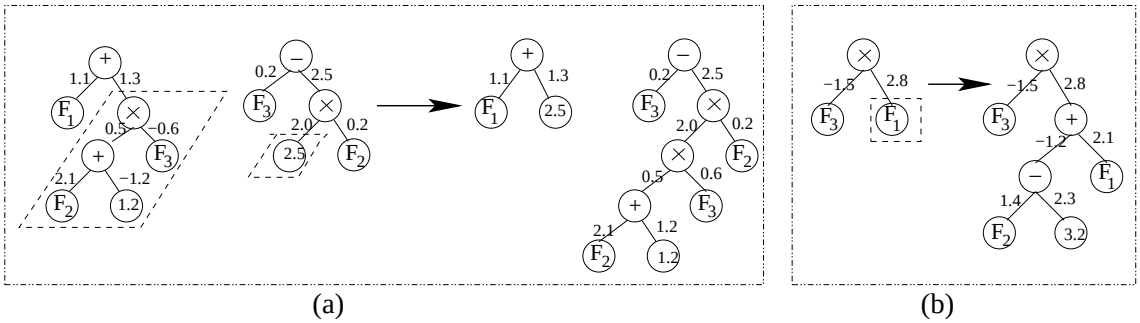


Figure 4: New genetic operators. (a) Crossover; (b) Mutation.

Based on this structure, changing a weight in a program will change the performance of the program. In this approach, all the weights are initialised to 1.0 as the standard GP approach, then are learned and updated through the gradient descent search algorithm. Note that the weight learning and updating process only happens locally for individual programs in a generation, and the genetic beam search still control the whole evolutionary process globally between different generations. We expect that this hybrid beam-gradient descent approach can improve the system performance.

3.2 Gradient Descent Applied to Program Weights

This section describes the gradient descent algorithm in this approach — how to apply gradient descent search to the weights in each program so that better programs can be found.

In this approach, gradient-descent is applied to changing the values of the weights. It is assumed that a continuous cost surface C can be found to describe the performance of a program at a particular classification task for all possible values for the weights. To improve the system performance, the gradient descent search is applied to taking steps "downhill" on the C from the current group of weights.

The gradient of C is found as the vector of partial derivatives with respect to the parameter values. This gradient vector points along the surface, in the direction of maximum-slope at the point used in the derivation. Changing the parameters proportionally to this vector (negatively, as it points to "uphill") will move the system down the surface C. If we use w_{ij}

to represent the value of the weight between node i and node j and y to represent the output of the genetic program P, then the distance moved (the change of w_{ij}) should therefore be:

$$\Delta w_{ij} = -\alpha \cdot \sum_{k=1}^{N} \frac{\partial C}{\partial w_{ij}} = -\alpha \cdot \sum_{k=1}^{N} \frac{\partial C}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{ij}} \tag{1}$$

where α is a search factor and N is the number of patterns in the training set. In the rest of this section, we will address the three parts — $\frac{\partial C}{\partial y}$, $\frac{\partial y}{\partial w_{ij}}$ and α.

3.2.1 Cost Surface and $\frac{\partial C}{\partial y}$

We used sum-squared error as the cost surface C, this is calculated as in equation 2.

$$C = \frac{\sum_{k=1}^{N} (y_k - Y_k)^2}{2} \tag{2}$$

where Y_k is the desired program output for training example k, y_k is the actual calculated program output for training example k, and N is the number of training examples.

Accordingly, the partial derivative of the cost function with respect to the genetic program would be:

$$\frac{\partial C}{\partial y_k} = \frac{\partial(\frac{(y_k - Y_k)^2}{2})}{\partial y_k} = y_k - Y_k \tag{3}$$

The corresponding desired output Y is calculated as follows:

$$Y_k = \textbf{class} - \frac{\textbf{numclass} + 1}{2} \tag{4}$$

where **class** is the class label of the object and **numclass** is the total number of classes. For example, for a five class problem as described in section 1, the desired outputs are $-2, -1, 0, 1$ and 2 for object classes 1, 2, 3, 4 and 5, respectively.

3.2.2 Partial Derivative $\frac{\partial y_j}{\partial x_i}$

For presentation convenience, y_k will be written as y with the pattern label k omitted.

In order to illustrate the calculation of the derivative of the program output by a change in the value of an inclusion factor, the program in figure 2 is used as an example.

If we use O_i to represent the output of node i, then the partial derivatives of the genetic program with respect to the weight w_{35} between node 3 and node 5 will be:

$$\begin{aligned} \frac{\partial y}{\partial w_{35}} &= \frac{\partial O_1}{\partial w_{35}} = \frac{\partial O_1}{\partial O_3} \cdot \frac{\partial O_3}{\partial w_{35}} \\ &= \frac{\partial(w_{12}O_2 + w_{13}O_3)}{\partial O_3} \cdot \frac{\partial(w_{34}O_4 \times w_{35}O_5)}{\partial w_{35}} \\ &= w_{13} \cdot w_{34} \cdot O_4 \cdot O_5 \end{aligned}$$

Since w_{13} and w_{34} are known and O_4 and O_5 can be obtained during evaluation of the program, the formula is readily calculated. In this way the appropriate derivative for any weight, in a program of any depth, can be obtained using the chain rule and the derivatives of the functions in the function set, as shown in table 2.

Table 2: Derivatives of the functions used in this approach.

Operation	$\frac{\partial o}{\partial a_1}$	$\frac{\partial o}{\partial a_2}$	$\frac{\partial o}{\partial w_1}$	$\frac{\partial o}{\partial w_2}$	$\frac{\partial o}{\partial a_3}$	$\frac{\partial o}{\partial w_3}$
$+$	w_1	w_2	a_1	a_2		
$-$	w_1	$-w_2$	a_1	$-a_2$		
$\times$	$w_1 w_2 a_2$	$w_2 w_1 a_1$	$a_1 w_2 a_2$	$a_2 w_1 a_1$		
$pdiv$	$\frac{w_1}{w_2 a_2}$	$-\frac{w_1 a_1}{a_2^2 w_2}$	$\frac{a_1}{w_2 a_2}$	$-\frac{w_1 a_1}{w_2^2 a_2}$		
if	0	if $(w_1 a_1 < 0)$ then $w2$ else 0	0	if $(w_1 a_1 < 0)$ then a_2 then 0	if $(w_1 a_1 < 0)$ then $w3$ else 0	$if(w_1 a_1 < 0)$ then a_3 else 0

3.2.3 Search Factor α

The search factor α in equation 1 was defined to be proportional to the inversed sum of the square gradients on all weights along the cost surface, as shown in equation 5.

$$\alpha = \eta \div \sum_{i=1}^{M} (\frac{\partial C}{\partial w_i})^2 \tag{5}$$

where M is the number of weights in the program, and η is a learning rate defined by the user. This equation was found to be adequate in previous research [15], and is reused here.

3.2.4 Summary of the Gradient Descent Algorithm

The gradient descent algorithm in this approach is summarised as follows.

- For each training pattern, evaluate the program, save the outputs of all nodes in the program.
- For each training pattern, calculate the partial derivative of the cost function on the program $\frac{\partial C}{\partial y}$ using equations 3 and 4.
- For each training pattern, calculate the partial derivatives of the program on each weights $\frac{\partial y}{\partial w_{ij}}$ using the chain rule and table 2.
- Calculate the search factor α using equation 5.
- Calculate the change of each weight using equation 1.
- Update the weights using $(w_{ij})_{new} = w_{ij} + \Delta w_{ij}$.

Note that this algorithm is an offline scheme, that is, the weights are updated after all the patterns in the training set are presented.

4 Data Sets

We used three data sets providing object classification problems of varying difficulty in the experiments. Example images are shown in figure 5.

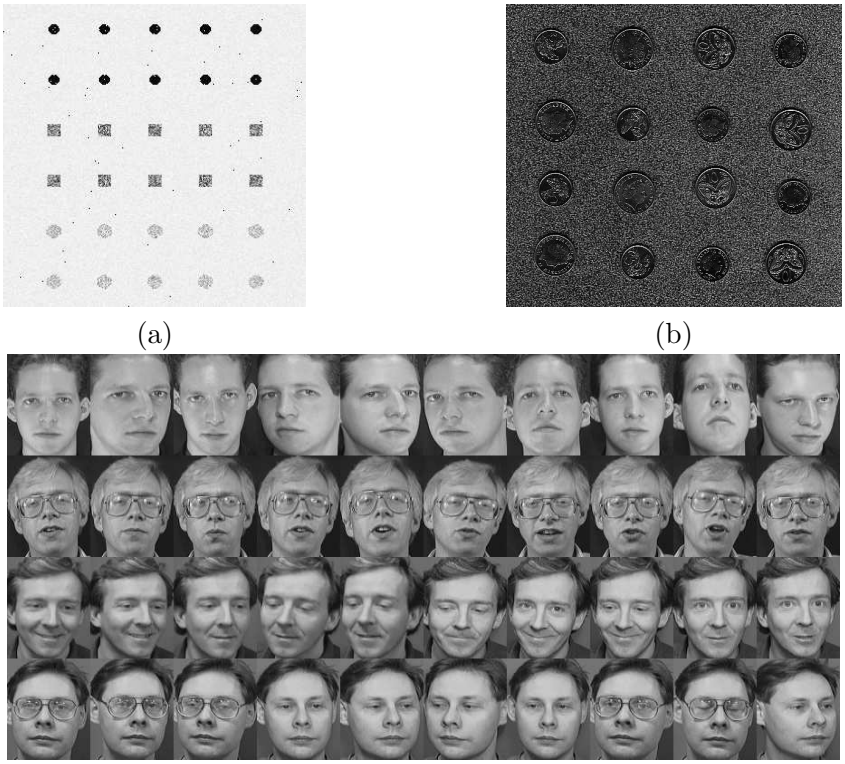


Figure 5: Sample image data sets. (a) Shape; (b) Coin; (c) Face.

4.1 Computer Generated Shape Data Set

The first set of images (figure 5a) was generated to give well defined objects against a relatively clean background. The pixels of the objects were produced using a Gaussian generator with different means and variances for each class. Three classes of 960 small objects were cut out from those images to form the classification data set. The three classes are: black circles, grey squares, and light circles. For presentation convenience, this dataset is referred to as *shape*.

4.2 NZ Coin Data Sets

The second set of images (figure 5b) contains scanned 5 cent and 10 cent New Zealand coins. The coins were located in different places with different orientations and appeared in different sides (head and tail). In addition, the background was cluttered. We need to distinguish different coins with different sides from the background. Five classes of 801 object cutouts were created: 160 5-cent heads, 160 5-cent tails, 160 10-cent heads, 160 10-cent tails, and the cluttered background (161 cutouts). Compared with the *shape* data set, the classification problem in this data set is much harder. Although these are still regular, man-made objects, the problem is very hard due to the noisy background and the low resolution.

4.3 Human Face Data Set

The third data set consists of 40 human faces (figure 5c) taken at different times, varying lighting slightly, with different expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no-glasses). These images were collected from the first four directories of the ORL face database [9]. All the images were taken against a dark homogeneous background

with limited orientations. The task here is to distinguish those faces into the four different people.

For the shape and the coin data sets, the objects were equally split into three separate data sets: one third for the training set used directly for learning the genetic program classifiers, one third for the validation set for controlling overfitting, and one third for the test set for measuring the performance of the learned program classifiers. For the faces data set, due to the small number of images, ten-fold cross validation was applied.

5 Experimental Results and Discussion

5.1 Experiment Configuration

The parameter values used in this approach are shown in table 3. The evolutionary process is run for a fixed number (*max-generations*) of generations, unless it finds a program that solves the classification perfectly (100% accuracy on training set), at which point the evolution is terminated early.

Table 3: Parameters used for GP training for the three datasets.

Parameter Names	Shape	coin	face	Parameter Names	Shape	coin	face
population-size	300	500	500	reproduction-rate	10%	10%	10%
initial-max-depth	5	5	5	crossover-rate	60%	60%	60%
max-depth	6	6	6	mutation-rate	30%	30%	30%
max-generations	100	100	100	cross-term	15%	15%	15%
object-size	16×16	70×70	92×112	learning rate η	1.0	1.0	1.0

5.2 Results

Table 4 shows the results of the new GP approach (GP-gradient) with different learning rates against the basic GP approach. The results at the learning rate "off" corresponds to the basic GP approach — no gradient descent search is used.

Table 4: A comparison of the results of the GP-gradient and the basic GP approach.

Dataset	Learning rate	Generations	Time (s)	Test Accuracy (%)
Shape	Off	15.56	0.85	99.66
	0.5	10.30	2.31	99.53
	1.0	6.38	1.39	99.69
	2.0	4.58	0.97	99.71
	3.0	5.52	1.22	99.80
Coin	0.0	43.46	1.68	90.89
	0.5	43.56	7.12	89.15
	1.0	43.34	7.48	90.22
	2.0	43.22	7.77	90.02
	3.0	43.10	7.53	92.14
Face	0.0	8.32	0.37	85.00
	0.5	8.07	0.35	86.50
	1.0	7.30	0.32	86.25
	2.0	6.23	0.27	86.30
	3.0	6.72	0.31	88.15

From these results, the following observations can be made:

- Different learning rates in the GP-gradient approach resulted in different performance. This is consistent with the nature of the gradient descent search — different local minima can be reached from different learning rates.
- For all the three data sets investigated here, it was always possible to find certain learning rates at which the GP-gradient approach outperformed the basic GP approach.
- For the three sets of experiments investigated here, it seemed that a learning rate of 3.0 achieved the best classification performance, suggesting that 3.0 could be used as a starting point.
- The new GP-gradient descent method almost always used fewer number of generations to find a good program classifier than the basic GP approach, and in some cases very much so. This suggests that the gradient descent search locally applied to individual programs did reduce the number of evolutions.
- In terms of the training time, it seems that the GP-gradient method spent more time than the basic GP approach in the easy shape data set and the coin data set. However, this is not the case for the relatively difficult face data set, where the new method used less training time to find a good program classifier.

Table 5 shows the results of this approach with the constraints of gradient descent search applied only to the weights connected with the numeric terminals. As shown in this table, the results is very similar to those obtained with gradient descent applied all weights connected with any sub-program tree classifier. This suggests that our previous approach, which applies gradient descent to the numeric terminal only, is a special case of the GP-gradient approach described in this paper, which can also achieve similar results on the same problems.

Table 5: Results of the approach with the constraints of gradient descent of numeric weights only.

Dataset	Learning rate	Generations	Time (s)	Test Accuracy (%)
Shape	Off	15.56	0.84	99.66
	0.5	10.06	2.33	99.74
	1.0	6.62	1.47	99.76
	2.0	5.96	1.35	99.71
	3.0	6.46	1.47	99.75
Coin	0.0	43.46	1.68	90.89
	0.5	43.42	7.13	89.64
	1.0	39.12	6.55	90.00
	2.0	38.94	6.53	91.87
	3.0	39.68	7.17	93.68
Face	0.0	7.31	0.13	84.15
	0.5	7.52	0.33	86.50
	1.0	7.11	0.31	86.75
	2.0	7.76	0.37	87.85
	3.0	7.46	0.35	88.65

6 Conclusions

This paper described an approach to the use of gradient descent search in tree based genetic programming for multiclass object classification problems. To learn better partial programs,

a weight parameter was introduced in each link between every two nodes in a program tree, so that a change of a weight corresponds to a change of the effect of the sub-program tree. Inside a particular generation, learning the changes of a weight parameter was done locally by gradient descent search, but the whole evolution process was still carried out across different generations globally by the genetic beam search. Unlike the previous approaches which also combines genetic beam search and gradient descent search but applies gradient descent to the numeric terminals/constants only, this approach can learn better program structures by updating the weights of the sub programs.

This approach was examined and compared with the basic genetic programming approach without gradient descent on three object classification problems of varying difficulty. The results suggest that the new approach outperformed the basic approach on all problems under certain learning rates.

The results also showed that different learning rates resulted in different performance. It was always possible to find certain learning rates, with which better performance could be achieved. It does not seem to have a reliable way to obtain a good learning rate, which usually needs an empirical search through experiments. However, if such a search could improve the system performance, such a short search is a small price to pay. According to our experiments, a learning rate of 3.0 seems a good starting points.

This work only used the offline learning scheme when updating the weights. We will investigate the effectiveness of the online scheme for updating the weights in this approach in the future.

References

[1] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, pages 477–494. MIT Press, 1994.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. San Francisco, Calif. : Morgan Kaufmann Publishers; Heidelburg : Dpunkt-verlag, 1998. Subject: Genetic programming (Computer science); ISBN: 1-55860-510-X.

[3] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in SAR imagery by genetic programming. *Advances in Engineering Software*, 30:303–311, 1999.

[4] John R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. Cambridge, Mass. : MIT Press, London, England, 1992.

[5] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, Mass. : MIT Press, London, England, 1994.

[6] Thomas Loveard and Victor Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP research group, editors, *Parallel distributed Processing, Explorations in the Microstructure of Cognition,*

Volume 1: Foundations, chapter 8. The MIT Press, Cambridge, Massachusetts, London, England, 1986.

[8] Conor Ryan and Maarten Keijzer. An analysis of diversity of constants of genetic programming. In E. Costa C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, editor, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 404–413, Essex, UK, 2003. Springer Verlag.

[9] F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *2nd IEEE Workshop on Applications of Computer Vision*, Sarasota (Florida), July 1994. ORL database is available at: www.cam-orl.co.uk/facedatabase.html.

[10] Andy Song, Vic Ciesielski, and Hugh Williams. Texture classifiers generated by genetic programming. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 243–248. IEEE Press, 2002.

[11] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[12] Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 330–335, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[13] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, pages 180–192, Sydney, Australia, December 1999. Springer-Verlag Berlin Heidelberg. Lecture Notes in Artificial Intelligence (LNAI Volume 1747).

[14] Mengjie Zhang, Victor Ciesielski, and Peter Andreae. A domain independent window-approach to multiclass object detection using genetic programming. *EURASIP Journal on Signal Processing, Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis*, 2003(8):841–859, 2003.

[15] Mengjie Zhang and Will Smart. Genetic programming with gradient descent search for multiclass object classification. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 399–408, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.