A Grammatical Genetic Programming Representation for Radial Basis Function Networks

Ian Dempsey[1], Anthony Brabazon[1], Michael O'Neill[1]

Natural Computing Research & Applications
University College Dublin
Ireland
ian.dempsey@gmail.com; anthony.brabazon@ucd.ie; m.oneill@ucd.ie

1 Introduction

General purpose neural network (NN) models such as multi-layer perceptrons (MLPs) and radial basis function networks (RBFNs) have been applied to many real-world problems. Although these models have very general utility, the construction of a quality network can be time consuming. Practical problems faced by the modeller include the selection of model inputs, the selection of model form, and the selection of appropriate parameters for the model such as weights. The use of evolutionary algorithms (EAs) such as the genetic algorithm provides scope to automate one or more of these decisions. Traditional methods of combining EA and NN methodologies typically entailed the encoding of aspects of the NN model using a fixed-length binary or real-valued chromosome. The EA is then applied to a population of chromosomes, each representing a specific NN structure. The population of chromosomes is evolved over time so that better NN structures are uncovered. A drawback of this method is that the use of a fixed length chromosome places a restriction on the nature of the NN models that can be evolved by the EA. This study adopts an alternative approach, using a novel hybrid algorithm where evolutionary computation, in the form of grammatical genetic programming, is used to generate an RBFN. This approach employs a variable length chromosome which implies that the structure of the RBFN is not determined *a priori* but rather is uncovered by means of an evolutionary process. This study represents the first application of a grammar-based genetic programming algorithm, namely Grammatical Evolution, to generate RBFNs.

In the remainder of this chapter the two components of the hybrid methodology are initially outlined (sections 2 and 3), followed by a description of how they are combined to form the hybrid algorithm (section 3). The results of the application of the hybrid algorithm to five benchmark classification problem

instances is provided in section 5. Conclusions and suggestions for future work are detailed in section 6.

2 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [12, 13, 14, 15, 16] and it can be considered a form of grammar-based genetic programming. GE has enjoyed particular success in the domain of Financial Modelling [2] amongst numerous other applications including Bioinformatics, Systems Biology, Combinatorial Optimisation and Design [11, 9, 4, 3]. Rather than representing the programs as parse trees, as in GP [5, 6, 1, 7, 8], a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. The grammar allows the generation of programs (or in this study, RBFN forms) in an arbitrary language that are guaranteed to be syntactically correct. As such, it is used as a generative grammar, as opposed to the classical use of grammars in compilers to check syntactic correctness of sentences. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar to produce very specific forms of sentences.

BNF is a notation that represents a language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of the set of terminals (the primitive symbols that can be used to construct the output program or sentence(s)), according to the production rules. A simple example of a BNF grammar is given below, where `<expr>` is the start symbol from which all programs are generated. These productions state that `<expr>` can be replaced with either one of `<expr><op><expr>` or `<var>`. An `<op>` can become either `+`, `-`, or `*`, and a `<var>` can become either `x`, or `y`.

```
<expr> ::= <expr><op><expr> (0)
         | <var>            (1)
  <op> ::= +                (0)
         | -                (1)
         | *                (2)
 <var> ::= x                (0)
         | y                (1)
```

The grammar is used in a developmental process to construct a program by applying production rules, selected sequentially using the genome, beginning from the start symbol of the grammar. In order to select a production rule in GE, the next codon value on the genome is read, interpreted, and placed in the following formula:

$$Rule = c \,\%\, r$$

where c is the codon value, r the number of choices for the current non-terminal, and % represents the modulus operator.

Fig. 1. An example GE individual's genome represented as integers for ease of reading.

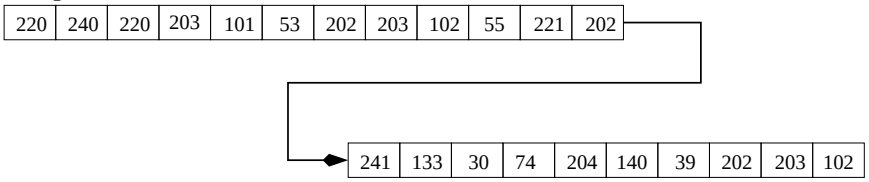

Fig. 1 provides an example of an individual genome (where each 8-bit codon is represented as an integer for ease of reading). The first codon integer value is 220, and given that we have 2 rules to select from for `<expr>` in the above grammar, we get 220 % 2 = 0. `<expr>` will therefore be replaced with `<expr><op><expr>`. Beginning from the the left hand side of the genome, codon integer values are generated and used to select appropriate rules for the left-most non-terminal in the developing program from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar. (b) The end of the genome is reached before the complete program is generated, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individuals mapping process. (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value.

Returning to the example individual, the left-most `<expr>` in `<expr><op><expr>` is mapped by reading the next codon integer value 240 and used in 240 % 2 = 0 to become another `<expr><op><expr>`. The developing program now looks like `<expr><op><expr><op><expr>`. Continuing to read subsequent codons and always mapping the left-most non-terminal the individual finally generates the expression `y*x-x-x+x`, leaving a number of unused codons at the end of the individual, which are deemed to be introns and simply ignored. A full description of GE can be found in O'Neill & Ryan (2003)[12]. Some more recent developments are covered in Brabazon & O'Neill (2005)[2].

3 Radial Basis Function Networks

A radial basis function network (RBFN) generally consists of a three-layer feedforward network. Like an MLP, RBFN can be used for prediction and classification purposes, but RBFNs differ from MLPs in that the activation functions of the hidden layer nodes are radial basis functions.

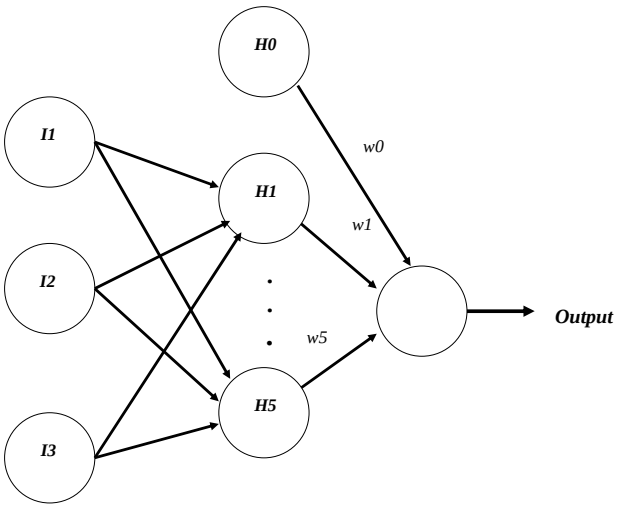

Fig. 2. A radial basis function network. The output from each hidden node (H0 is a bias node, with a fixed input value of 1) is obtained by measuring the distance between each input pattern and the location of the hidden node, and applying the radial basis function to that distance. The final output from the network is obtained by taking the weighted sum (using w0, w1 and w5) of the outputs from the hidden layer and from H0

The training of RBFNs typically consists of a combination of unsupervised and supervised learning. Initially, a number hidden layer nodes (or *centres*) must be positioned in the input data space. This can be performed by following a simple rule, or in a more sophisticated application by using unsupervised learning. Methods for choosing the locations of centers include distributing the centres in a regular grid over the input space, selection of a random subset of the training data vectors to serve as centres, or using an algorithm to cluster the input data (e.g. SOMs can be used for this) and then selecting a centre location to represent each cluster. Each of these centres forms a hidden node in the RBFN's structure.

Input data vectors are typically standardised before training. When each input vector is presented to the network a value is calculated at each centre

using a radial basis function. This value represents the quality of the match between the input vector and the location of that centre in the input space. Each hidden node, therefore, can be considered as a local detector in the input data space. The most commonly used radial basis function is a Gaussian function. This produces an output value of one if the input and weight vectors are identical, falling towards zero as the distance between the two vectors gets large. A range of alternative radial basis functions exists, including the inverse multi-quadratic function and the spline function.

The second phase of the model construction process is the determination of the value of the weights on the connections between the hidden layer and the output layer. In training these weights, the output value for each input vector will be known, as will the activation values for that input vector at each hidden layer node, so a supervised learning method can be used. The simplest transfer function for the node(s) in the output layer is a linear function where the network's output is a linearly weighted sum of the outputs from the hidden nodes. In this case, the weights on the arcs to the output node(s) can be found using linear regression, with the weight values being the regression coefficients. Sometimes it may be preferred to implement a non-linear transfer function at the output node(s). For example, when the RBFN is acting as a binary classifier it would be useful to use a sigmoid transfer function to limit outputs to the range $0 \rightarrow 1$. In this case, the weights between the hidden and output layer could be determined using the backpropagation algorithm.

Once the RBFN has been constructed using a training set of input-output data vectors it can then be used to classify or to predict outputs for new input data vectors, for which an output value is not known. The new input data vector is presented to the network, and an activation value is calculated for each hidden node. Assuming that a linear transfer function is used in the output node(s), the final output produced by the network is the weighted sum of the activation values from the hidden layer, where these weights are the coefficient values obtained in the linear regression step during training. The basic algorithm for the canonical RBFN is as follows:

i. Select the initial number of centres (m).
ii. Select the initial location of each of the centres in the data space.
iii. For each input data vector/centre pairing calculate the activation value $\phi(||x - y||)$, where ϕ is a radial basis function and $||...||$ is a distance measure between input vector x and a centre y in the data space. As an example, let $d = ||x - y||$. The value of a Gaussian RBF is then given by $y = \exp(\frac{-d^2}{2\sigma^2})$, where σ is a modeller selected parameter which determines the size of the region of input space a given centre will respond to.
iv. Once all the activation values for each input vector have been obtained, calculate the weights for the connections between the hidden and output layers using linear regression.
v. Go to step (iii) and repeat until a stopping condition is reached.

vi. Improve the fit of the RBFN to the training data by adjusting some or all of the following: the number of centres, their location, or the width of the radial basis functions.

As the number of centres increases, the predictive ability of the RBFN on the training data will tend to increase, possibly leading to overfit and poor out-of-sample generalisation. Hence, the object is to choose a sufficient number of hidden layer nodes to capture the essential features in the training data, without overfitting it.

4 GE-RBFN Hybrid

Despite the apparent dissimilarities between GE and RBFN methodologies, the methods can complement each other. A practical problem in utilising RBFNs is the selection of model inputs and model form. By defining an appropriate grammar, GE is capable of automatically generating a range of RBFN forms. Hence, a combined GE-RBFN hybrid can be considered as embedding both hypothesis generation and hypothesis optimisation components.

The basic operation of the GE-RBFN methodology is as follows. Initially, a population of binary strings are randomly created. In turn, each of these is mapped to a RBFN structure using a grammar which has been constructed specifically for the task of generating RBFNs (see next subsection). The quality of each resulting RBFN is then assessed using the training data. Based on this information, the binary strings resulting in higher quality networks are preferentially selected for survival and reproduction. Over successive iterations, the quality of the networks encoded in the population of binary strings improves.

4.1 Grammar

There are multiple grammars that could be defined in order to generate RBFNs depending on exactly what the modeller wishes to evolve. For example, if little was known about which inputs would be useful for the RBFN, the grammar could be written so that GE selected which inputs to use, in addition to selecting the form of the RBFN itself (for example, the number of hidden layer nodes, their associated weight vectors, the form of their associated radial basis functions and so on).

In this study we define a grammar which permits GE to construct RBFNs with differing numbers of centres. GE is also used to decide where to locate those centres in the input space. The Backus Naur Form grammar for this is as follows.

```
<RBFN> :: = 1 / (1 + exp (- <HL>) )

<HL> ::= <weight> * <HN>
```

```
        | <weight> * <HN> + <HL>

<HN> ::= <gaussian>

<center> ::= <real>, <real> (one item for each

<radius> ::= <real>

<weight> ::= <real>

<real> ::=  your constant generation method of choice
```

where the non-terminal <gaussian> ::= $exp^{\frac{-\sum_{i=1}^{V}(input[I][i]-center[HN][i])^2)}{2*(<radius>^2)}}$.

Under the above grammar, the generation of a RBFN starts from the root $< RBFN >$. This can only be mapped to one choice, hence it gives rise to the expression $1/(1+exp(- < HL >))$. Next, the non-terminal in this expression $< HL >$ is mapped into either $< weight > * < HN >$ or $< weight > * < HN > + < HL >$, depending on the value of the next codon on the binary genome. Suppose the next codon on the genome gives rise to an integer value of 34. Taking 34 Mod 2 (the number of choices available for $< HL >$) gives 0, hence $< HL >$ becomes the first choice, $< weight > * < HN >$. At this point, the RBFN consists of a network with a single hidden layer node. In subsequent derivation steps, the real numbers corresponding to the location of this centre, and the real number corresponding to the radius of the centre are derived, eventually giving rise to a complete RBFN form.

4.2 Example Individuals

Fig. 3 provides a graphical illustration of the possible derivation trees which the grammar could create. Tree **A** illustrates the basic form that all the RBFN will take. The non-terminal <HL> is then expanded and can result in a RBFN which has one or more hidden layer nodes. The RBFN generation process iterates until all the non-terminals are mapped to terminals.

5 Experimental Setup & Results

Five benchmark classification problem instances from the UCI Machine Learning Repository [17] are tackled. Summary statistics on each problem instance are provided in Table 1. Each dataset was recut between training and test data ten times, with 80% of the dataset being used for training and 20% for out of sample testing in each case. In assessing the quality of the developed RBFNs, the number of correct classifications produced was used.

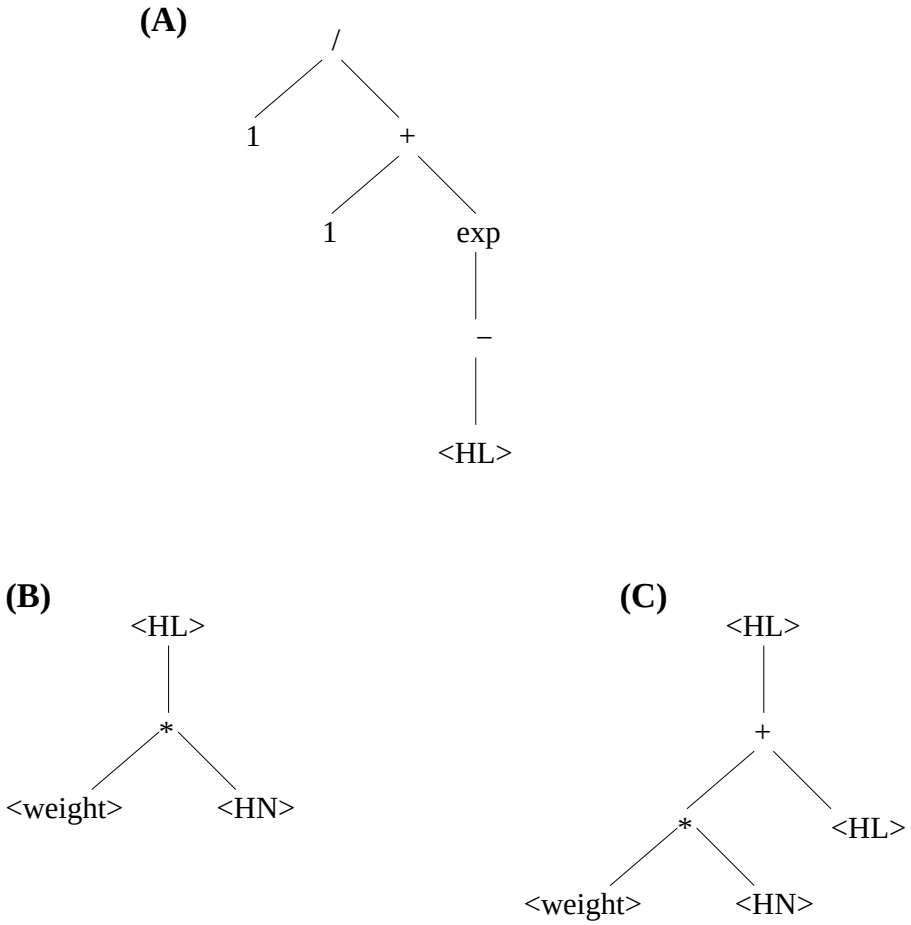

Fig. 3. An output radial basis function network in the form of a derivation tree. Tree **(A)** represents the common structure of all RBFN's generated by the example grammar. Trees **(B)** and **(C)** represent the two possible sub-trees that can replace the <HL> non-terminal in **(A)**. **(B)** represents the case where a <HL> becomes a single node, and **(C)** represents the case where <HL> becomes at least two nodes.

The Wisconsin problem is a data set of malignant and benign breast cancer cases. Pima includes data on Pima Indians Diabetes from the National Institute of Diabetes and Digestive and Kidney Diseases. The Thyroid data set is made up of thyroid patient records classified into disjoint disease classes. Australian data set is made up of cases of credit card applications from the Credit Screening Database and the Bupa data set is from Bupa Medical Research Ltd. and has data on liver disorders.

Table 1. Problem instance statistics and the training and test set partition sizes in each case.

Dataset	Training	Test	#variables	#classes
Wisconsin	559	140	9	2
Pima	614	154	8	2
Thyroid	172	43	5	3
Australian	552	138	6	2
Bupa	276	69	6	2

The results obtained by the hybrid system over the ten recuts are reported in Table 2. Overall the results are encouraging. Comparing them against previously published results from [18] on four of the same datasets (see Table 3), it can be seen that the evolved RFBNs outperform on two of the datasets, and underperform on the other two.

It should be noted that there is considerable room to fine-tune the parameters of the GE-RBFN hybrid, and this provides scope to further improve the above results. In this proof of concept study, typical off-the-shelf parameter settings were adopted for GE. A population size of 500 individuals was used with 100 generations of training. A generational rank replacement strategy was used with 25% of the weakest performing members of the population being replaced with newly generated individuals on each generation. For each dataset, a total of 30 runs conducted with a crossover rate of 0.9 and a mutation rate of 0.1 as in [19]. All reported results are averaged over the 30 runs.

Table 2. Results for GE/RBFN including average fitnesses for both in and out of sample data sets along with standard deviation for the out of sample data.

	Mean best in sample	Mean best out of sample	Std. dev.
Australian	70.52	71.53	4.059
Bupa	60.26	57.11	4.504
Thyroid	62.40	75.78	4.559
Wisconsin	88.92	95.20	2.643
Pima	68.82	67.53	3.647

6 Conclusions & Future Work

This study presents a novel approach, based on a form of grammatical genetic programming (grammatical evolution), for the automatic generation of RBFNs. A particular feature of this methodology is that the structure of the

Table 3. Comparative out of sample results.

	Mean best out of sample	**Std. dev.**
Bupa	65.97	11.27
Thyroid	96.27	4.17
Wisconsin	95.63	1.58
Pima	73.50	4.23

resulting RBFN is not defined *a priori*, but is evolved during the construction process. The developed GE-RBFN hybrid was applied to five benchmark instances from the UCI Machine Learning repository with encouraging results.

Substantial scope exists to further develop the RBFN-GE hybrid outlined in this chapter. In this initial study we did not include the selection of inputs, or the selection of the form of the RBFs in the evolutionary process. However, the RBFN grammar could be easily adapted in order to incorporate these steps if required. The use of the GE methodology also opens up a variety of other research avenues. The GE methodology applied in this study is based on a canonical form of the GE algorithm. As already noted, a substantial literature exists on GE, covering such issues as the use of alternative search engines for the algorithm, and the use of alternatives to the strict left-to-right mapping of the genome (piGE). Future work could usefully examine the utility of these GE variants for the purposes of evolving RBFNs.

References

1. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann.
2. Brabazon, A., O'Neill, M. 2005. *Biologically Inspired Algorithms for Financial Modelling*. Springer.
3. Cleary, R., O'Neill, M. 2005. An Attribute Grammar Decoder for the 01 Multi-Constrained Knapsack Problem. In LNCS 3448 *Proc. of Evolutionary Computation in Combinatorial Optimization EvoCOP 2005*, pp.34-45, Lausanne, Switzerland. Springer.
4. Hemberg, M., O'Reilly, U-M. 2002. GENR8 - Using Grammatical Evolution In A Surface Design Tool. In *Proc. of the First Grammatical Evolution Workshop GEWS2002*, pp.120-123. New York City, New York, US. ISGEC.
5. Koza, J.R. (1992). *Genetic Programming*. MIT Press.
6. Koza, J.R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.
7. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann.
8. Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

9. Moore, J.H., Hahn, L.W. (2004). Systems Biology Modeling in Human Genetics Using Petri Nets and Grammatical Evolution. In LNCS 3102 *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2004*, Seattle, WA, USA, pp.392-401. Springer.
10. O'Neill, M., Brabazon, A. (2004). Grammatical Swarm. In LNCS 3102 *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2004*, Seattle, WA, USA, pp.163-174. Springer.
11. O'Neill, M., Adley, C., Brabazon, A. (2005). A Grammatical Evolution Approach to Eukaryotic Promoter Recognition. In *Proc. of Bioinformatics INFORM 2005*, Dublin City University, Dublin, Ireland.
12. O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.
13. O'Neill, M. (2001). *Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution*. PhD thesis, University of Limerick, 2001.
14. O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation*. 2001.
15. O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, Vol. 4 No. 1. Kluwer Academic Publishers, 2003.
16. Ryan, C., Collins, J.J., O'Neill, M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Proc. of the First European Workshop on GP*, 83-95, Springer-Verlag.
17. Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.
18. Smith, M. and Bull, L. (2005). Genetic Programming with a Genetic Algorithm for Feature Construction and Selection, *Genetic Programming and Evolvable Machines*, 6(3):265-281.
19. Dempsey, I., O'Neill, M. and Brabazon, A. (2002). Investigations into Market Index Trading Models Using Evolutionary Automatic Programming, In *LNAI 2464, Proceedings of the 13th Irish Conference in Artificial Intelligence and Cognitive Science*, pp. 165-170, edited by M. O'Neill, R. Sutcliffe, C. Ryan, M. Eaton and N. Griffith, Berlin: Springer-Verlag.