Neural Network Synthesis via Asynchronous Analytic Programming

PAVEL VAŘACHA
Tomas Bata University in Zlin
Faculty of Applied Informatics
nam. T.G. Masaryka 5555, 760 01 Zlin
CZECH REPUBLIC
varacha@fai.utb.cz  http://www.fai.utb.cz

Abstract: This article deals with Analytic Programming (AP) which was proven to be highly effective tool of Artificial Neural Network (ANN) synthesis and optimization. New innovative asynchronous distribution of Self-Organizing Migration Algorithm (SOMA) is introduced and used together with AP. Such implementation can for example save 67% of computation time if distributed between 8 processors. Efficiency of AP as well as asynchronous distribution of SOMA was tested and statistically measured on 921937 evaluations, each of them containing another separate execution of SOMA.

Key-Words: Neural Network, Analytic Programming, SOMA, optimization, parallel evolutionary algorithm

1 Introduction
This article describes process of Artificial Neural Network (ANN) synthesis via symbolic regression. There are two well-known methods: Genetic Programming and Grammatical Evolution, which can both symbolically regress using evolutionary algorithm. However, in this article, more recent and flexible procedure called Analytic Programming (AP) is used here.

AP performed well in many separate cases (for example [1, 2]) together with different evolutionary algorithms (EA) as its “engine”. Asynchronous implementation of SOMA – Self-Organizing Migration Algorithm [3] is applied here to boost AP with the ambition to show unusual electivity of such arrangement.

SOMA is based on a self-organizing behavior of groups of individuals in a “social environment”. It can also be classified as an evolutionary algorithm [4], despite the fact that no new generations of individuals are created during the search (due to philosophy of this algorithm). Only positions of individuals in the searched space are changed during one generation, called a “migration loop”. The algorithm was published in journals, book and presented at international conferences, symposiums as well as in various invitational presentations, for example [5, 6, 7].

“Disadvantage” of SOMA, as well as of others evolutionary algorithms is their complicated and often ineffective parallelism. [3] Nevertheless, as the article describes further on, unique features of SOMA enable optimized distribution of heuristic computations and thereby fast and fruitful ANN synthesis. The whole process is theoretically explained and statistically proved below.

2 Analytic Programming
Main principle (core) of AP is based on discrete set handling (DSH) (Fig. 1). DSH shows itself as universal interface between EA and a symbolically solved problem. This is why AP can be used almost by any EA.

![Fig. 1, DSH principle](image)

Briefly put, in AP, individuals consist of non-numerical expressions (operators, functions,…), which are within evolutionary process represented by their integer indexes. Such index then serves like a pointer into the set of expressions and AP uses it to synthesize resulting function-program for Cost Function evaluation.

All simple functions and operators are in so called General Function Set (GFS) divided into groups according to the number of arguments which
can be inserted during the evolutionary process to create subsets $\text{GFS}_{3\text{arg}}$, $\text{GFS}_{2\text{arg}}$, ..., $\text{GFS}_{0\text{arg}}$.

Fig. 2, Example of GFS and its subsets

Functionality of discrete set handling can be seen on the concrete example in Fig. 3:

The individual consists of 6 arguments (indices, pointers to GFS). The first index is 1 meaning that $+$ is taken from the set of functions $\text{GFS}_{\text{all}}$. Function $\text{plus}$ has two arguments, therefore indexes 6 and 7 are arguments of $\text{plus}$.

\[
6 + 7
\]

Index 6 is then replaced by $\text{Sin}$ and index 7 by $\text{Cos}$.

\[
\text{Sin} + \text{Cos}
\]

$\text{Sin}$ and $\text{Cos}$ are one-argument functions. Then, index 7 follows index 8, which is replaced by $\text{Tan}$.

\[
\text{Sin}(\text{Tan}) + \text{Cos}
\]

$\text{Tan}$ is also one-argument function. Then, after index 8 the individual takes index 9, which is replaced by $t$ and this $t$ becomes the argument of $\text{Cos}$.

\[
\text{Sin}(\text{Tan}) + \text{Cos}(t)
\]

But in our case there is a function $\text{Mod}$. $\text{Mod}$ needs an argument to work properly. AP will not allow this, as there is not any other free pointer to be used with the argument. Instead of $\text{Mod}$, the AP will jump into the subspace, in this case directly to $\text{GFS}_{0\text{arg}}$. In the $\text{GFS}_{0\text{arg}}$ it finds 11th element which is $t$. And by doing so, we get (5) [1].

\[
\text{Sin}(\text{Tan}(t)) + \text{Cos}(t)
\]

Number of actually used pointers from an individual before synthesized expression is closed is called depth.

2.1 Constant Processing

Synthesized ANN, programs or formulas may also contain constants “K” which can be defined in $\text{GFS}_{0\text{arg}}$ or be a part of other functions included in $\text{GFS}_{\text{all}}$. When the program is synthesized, all Ks are indexed, so $K_1$, $K_2$, ..., $K_n$ are obtained and then all $K_n$ are estimated by second EA. In this case the EA is, again, asynchronous implementation of SOMA.

This is especially convenient for an ANN synthesis. $K_n$ can be interpreted as various weights and thresholds and their optimization by SOMA as ANN learning.

3 SOMA All-to-One

Several different versions of SOMA exist, nevertheless, this article is focused on most common All-to-One version, which is most suitable for asynchronous parallel implementation. All basic All-to-One SOMA principles important for correct understanding of executed experiment are described below.

3.1 Parameter definition

Before starting the algorithm, SOMA’s parameters: Step, PathLength, PopSize, PRT and Cost Function need to be defined. The Cost Function is simply the function which returns a scalar that can directly serve as a measure of fitness. In this case, Cost Function is provided by AP.

3.2 Creation of Population

Population of individuals is randomly generated. Each parameter for each individual has to be chosen randomly from the given range $<\text{Low}, \text{High}>$.

3.3 Migration loop

Each individual from population (PopSize) is evaluated by the Cost Function and the Leader (individual with the highest fitness) is chosen for the
current migration loop. Then, all other individuals begin to jump, (according to the Step definition) towards the Leader. Each individual is evaluated after each jump by using the Cost Function. Jumping continues until a new position defined by the PathLength is reached. The new position \( x_{i,j} \) after each jump is calculated by (6) as is shown graphically in Fig. 1. Later on, the individual returns to the position on its path, where it found the best fitness.

\[
X_{i,j}^{new} = X_{i,j,\text{start}}^{ML} + (X_{i,j-1}^{ML} - X_{i,j,\text{start}}^{ML}) t \text{ PRTVector}_j
\]

where \( t \in <0, \text{by Step to, PathLength}> \)
and ML is actual migration loop

Before an individual begins jumping towards the Leader, a random number \( r_d \) is generated (for each individual’s component), and then compared with PRT. If the generated random number is larger than PRT, then the associated component of the individual is set to 0 using PRTVector.

\[
\text{if } r_d < \text{PRT then PRTVector}_j = 0 \text{ else } 1
\]

where \( r_d \in <0, 1> \)
and \( j = 1, \ldots, n_{\text{param}} \)

<table>
<thead>
<tr>
<th>( j )</th>
<th>( r_d )</th>
<th>PRTVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.234</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.545</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.865</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.012</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1, An example of PRTVector for 4 parameters individual with PRT = 0.3

Hence, the individual moves in the N-k dimensional subspace, which is perpendicular to the original space. This fact establishes a higher robustness of the algorithm. Earlier experiments demonstrated that without the use of PRT, SOMA tends to determine a local optimum rather than global one. [8]

3.4 Test for stopping condition
If a divergence between current Leader and Leader from the last migration loop is less than defined number, stop and recall the best solution(s) found during the search.

\[
\text{Fig. 4, PRTVector and its action on individual movement [8]}
\]

4 Neural Network Synthesis
There is a very easy way of using AP for ANN synthesis. [9] The most important part is to define items from which ANN will be composed. In this case GFS contains only three items.

\[
GFS_{dil} = \{+, \text{AN}, K*x\}
\]

Most important item of (8) is Artificial Neuron (AN) (9) with hyperbolic tangent as transfer function (10). Weight of output, steepness and thresholds are computed as K in AP (11).

\[
GFS_1 = \{\text{AN}\}
\]

\[
AN = w \ast (e^{\lambda \ast (input + \phi)} - 1) / (e^{\lambda \ast (input + \phi)} + 1);
\]

\[
AN = K_1 \ast (e^{\lambda \ast (input + K_2)} - 1) / (e^{\lambda \ast (input + K_2)} + 1);
\]

\[
\text{Fig. 5, Graphical example of AN}
\]

To allow more inputs into one ANN simple plus operator (12) is used.
Finally, (13) represents weighted input data.

\[ GFS_0 = K^* x \]  

Fig. 7, Graphical example of weighted input

Under such circumstances, translation of an individual to ANN can be easily grasped from Fig. 8.

Whole process is cyclical. Individuals provided by EA are translated to ANNs. ANNs are evaluated in accordance with training data set and their global errors are used to set fitness to these individuals. Consequently, a new generation is chosen and the whole process is repeated in next migration loop.

5 Asynchronous SOMA distribution

Chapter 4 explains the process during which huge number of very different ANNs can be synthesized. Therefore, an actual population, which needs to be evaluated contains individuals with various number of \( K_n \). This means that the algorithm is very time demanding and furthermore, computation of every individual consumes different computation time. [3]

Fortunately, in these days, standard computers are much more often equipped by more than one processor. However, if the individuals are evenly divided between available processors for every migration loop, huge amount of computation time is lost due their unevenly distributed complexity.

To overcome this set-back, small but very important change to SOMA mechanism was made. The individuals no longer work in migration loops (see, chapter 3.3). On the contrary:

Every individual is compared with the Leader just after it finishes its jumping and new Leader is selected immediately.

This makes SOMA distribution work asynchronously. All the individuals do their migrations independently and some may even move much faster than others.

As there is not more any synchronization point to evaluate stopping condition (chapter 3.4), the condition is evaluated once after \( n \) evaluations of Cost Function, \( n = \text{period} \times \text{number of individuals} \times \text{mass} / \text{step} \).

The experiment described bellow is aimed to statistically show how this breakthrough asynchronous SOMA distribution can save computation time, work quickly and produce very good results of ANN optimization.

3.1 Reinforced evolution

If ANN of adequate quality cannot be obtained during AP run, AP puts the best ANN it found as a sub ANN into \( GFS_0 \) and starts over. This arrangement considerably improves AP ability to find ANN with desirable parameters.

6 Experiment set up

The function (14) was chosen to be regressed by ANN.

\[ y = x_i^5 - 2 x_i^3 + x_i \]

where \( x_i \in \langle -1, \text{by the step} 0.04, 1 \rangle \)
AP was executed 100 times (physically on 8 Intel(R) Core(TM) i7 3.19 GHz processors) to produce ANN with Root Mean Square Deviation (15) \( \text{RMSD} < 0.005 \). Main intention was to find such ANN, which meets this condition and which simultaneously used as few AN as possible.

\[
\text{RMSD} (\theta_1, \theta_2) = \sqrt{\frac{\sum_{i=1}^{n} (ANN (x_i) - y (x_i))^2}{n}}
\]

Setting of Asynchronous SOMA used as EA for AP can be seen in table 2.

<table>
<thead>
<tr>
<th>Number of Individuals</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual Parameters</td>
<td>100</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>High</td>
<td>3</td>
</tr>
<tr>
<td>PathLength</td>
<td>3</td>
</tr>
<tr>
<td>Step</td>
<td>0.11</td>
</tr>
<tr>
<td>PRT</td>
<td>1/depth</td>
</tr>
<tr>
<td>Divergence</td>
<td>0.01</td>
</tr>
<tr>
<td>Period</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2, Setting of SOMA used as EA for AP

Table 3, Setting of SOMA used to optimize \( K_n \)

<table>
<thead>
<tr>
<th>Number of Individuals</th>
<th>number of ( K_n ) * 0.5 (at least 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual Parameters</td>
<td>100</td>
</tr>
<tr>
<td>Low</td>
<td>-10</td>
</tr>
<tr>
<td>High</td>
<td>10</td>
</tr>
<tr>
<td>PathLength</td>
<td>3</td>
</tr>
<tr>
<td>Step</td>
<td>0.11</td>
</tr>
<tr>
<td>PRT</td>
<td>1/ number of ( K_n )</td>
</tr>
<tr>
<td>Divergence</td>
<td>0.01</td>
</tr>
<tr>
<td>Period</td>
<td>6</td>
</tr>
</tbody>
</table>

All 100 AP runs successfully synthesized ANN with RMSD < 0.005 in average number of used AN was 9. Nevertheless, the optimization task to find ANN with lowest number of AN was performed best in 4 cases, which employed only 2 AN. All this cases lead to similar ANN structure.

7 Results

In total 921937 evaluations of AP individual fitness was done during 100 AP executions and separated SOMA run was performed for all of them to set theirs \( K_n \). Time needed for all these evaluations was approximately 5 hours and 24 minutes.

Average time of 1 evaluation was 558 ms however \( t_{\text{max}} = 136369 \) ms at that 98% of measured times \( t < t_{\text{max}} / 10 \). Such results prove that vast amounts of computation time can be saved by asynchronous distribution (see chapter 5). How these values increase with increasing number of processors used is described in table 5 and Fig. 9.

Table 4, Time saved by asynchronous evaluation

<table>
<thead>
<tr>
<th>number of processors used</th>
<th>percentage of saved time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>23.7 %</td>
</tr>
<tr>
<td>4</td>
<td>47.3 %</td>
</tr>
<tr>
<td>8</td>
<td>67.2 %</td>
</tr>
<tr>
<td>16</td>
<td>81.2 %</td>
</tr>
</tbody>
</table>

Fig. 9, Time saved by asynchronous evaluation

Example of a successfully optimized ANN (17) and its sub ANN (16):

\[
ANN0 = x + AN[x]
\]  

(16)

\[
ANN1 = ANN0 + AN[ANN0] + ANN0
\]  

(17)

After successful optimization of \( K_n \) by SOMA (16), (17) lead to (18), (19).

\[
ANN0 = -0.972628914257888 * x + 0.960043432203328 * AN[0.303565531015147 * (7.00172920571721 + -0,00454216333835794)]
\]  

(18)
ANN described as (16), (17) can be graphically interpreted (Fig 11).

\[ \text{ANN} = \text{ANN}_0 + 0.40897485611192 \times \text{AN}[-2.77100775198393 \times (\text{ANN}_0 + 0.00305134718869929)] + \text{ANN}_0 \]  

Fig 11, Graphical interpretation of resulted ANN

8 Conclusion
Asynchronous distributions proved itself to be crucially important to successful AP implementation. For example, if 8 processors are used (as they were in the experiment), more than 67% of computation time (which would be wasted otherwise) can be saved. Considering the experiment, around 3 hours of computation time were saved.

AP also exercised ability to synthesized ANN affectively and with use of asynchronous SOMA distribution also quickly (0.5 s for 1 ANN in average).

Very small ANN containing only two ANs was automatically found and solved given problem with satisfactory RMSD. This success shows AP as a very useful tool of ANN synthesis and optimization.

9 Acknowledgment
The work was performed with financial support of research project NPVII-2C06007, by the Ministry of Education of the Czech Republic.

References:

978-960-474-292-9 97


