ESANN'2007 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2007, d-side publi., ISBN 2-930307-07-2.

Adaptive Global Metamodeling with Neural Networks

Dirk Gorissen, Wouter Hendrickx, Tom Dhaene

University of Antwerp - Department of Math and Computer Science
Middelheimlaan 1, 2000 Antwerp, Belgium

Abstract. Due to the scale and computational complexity of current simulation codes, metamodels (or surrogate models) have become indispensable tools for exploring and understanding the design space. Consequently, there is great interest in techniques that facilitate the construction and evaluation of such approximation models while minimizing the computational cost and maximizing metamodel accuracy. This paper presents an adaptive, integrated approach to global metamodeling based on the Multivariate Metamodeling Toolbox. An adaptive, evolutionary inspired, modeling algorithm based on neural networks is presented and its performance compared with rational metamodeling on a number of test problems.

1 Introduction

For many problems from science and engineering it is impractical to perform experiments on the physical world directly (e.g., airfoil design, earthquake propagation). Instead, complex, physics-based simulation codes are used to run experiments on computer hardware. However, due to the computational complexity of such high-fidelity codes, design space exploration or parameter optimization quickly becomes intractable [7]. As a result the use of approximation models (also known as metamodels, surrogate models and response surface models) that mimic the behavior of the simulation model (from now on referred to as the *simulator*) has become commonplace. Examples of metamodels include: RBF functions, Kriging models, Support Vector Machines (SVM) and Artificial Neural Networks (ANN).

Before we continue we first need to make a very important distinction between two different applications of metamodels. The first is by far the most popular and involves building small, simple metamodels for use in design optimization. Simple metamodels are used to guide the search towards a global optimum [7]. In the second case one is not interested in finding the optimal parameter vector but rather in the global behavior of the system. Here the metamodel is tuned to mimic the underlying model as closely as needed on a subset of the domain. In this paper we are concerned with the latter.

The remainder of the paper is structured as follows: in the next section we describe the research platform we have used to conduct our experiments. Section 3 gives an overview of related work, followed by the treatment of the neural network modeler used to exemplify our approach in section 4. Then we test the performance of the modeler on a number of test problems and compare the results to rational modeling in section 5. We conclude the paper with an evaluation of the results in section 6.

2 M3-Toolbox

The principal reason for turning towards approximation models is that the simulator is too time consuming to run for a large number of simulations. Nevertheless, one could

argue that in order to obtain an accurate global metamodel one still needs to perform numerous simulations, thus running into the same problem. However, we argue that this is not the case since: (1) building a metamodel is a one-time, up-front investment, (2) distributed computing can speedup the evaluation time and (3) adaptive modeling and adaptive sampling (sequential design) can drastically decrease the required datapoints to produce a good model. We have integrated (2) and (3) into a common research platform: the Matlab **M**ultivariate **M**eta**M**odeling Toolbox (M3-Toolbox).

The M3-Toolbox was developed when research made clear that there was room for an adaptive tool that integrated different modeling approaches and did not tie the user down to one particular set of problems. More concretely, we were interested in a fully automated, adaptive global metamodel construction algorithm. Given a simulation model the software should produce a metamodel with as little user interaction as possible. However, at the same time keeping in mind that there is no such thing as a 'one-size-fits-all', different problems need to be modeled differently. Therefore the software should be modular and extensible but not be too cumbersome to use or configure. In sum, the emphasis of the toolbox lies on the different levels of pluggability: model types (rational functions, ANN, SVM, ...), modeling algorithms (sequential, genetic, ...), sample selection (random, error based, density based, ...), model selection (cross validation, AIC, ...) and sample evaluation (local, on a cluster or grid). The behavior of each component is configurable through a central XML configuration file and components can easily be added, removed or replaced by custom implementations.

In short, the working of the toolbox is as follows: an initial set of samples (data points) is chosen, one or more surrogate models are constructed and their parameters varied according to some modeling algorithm (e.g., a genetic algorithm). Models are assigned a score based on one or more measures (e.g., cross validation, AIC) and the adaptive modeling continues until no further improvement is possible. The models are then ranked according to their score and new samples are selected based on the top k models. The adaptive modeling resumes and the whole process repeats itself until a threshold has been exceeded or the user required accuracy has been reached. Please refer to [4] for further information on the toolbox architecture and control flow. The toolbox, including all test problems, is freely available for non-commercial use at `http://www.coms.ua.ac.be`.

3 Related Work

Regression metamodels have found their way into virtually every scientific field. Neural networks have been particularly ubiquitous with successful applications in, among many others, economics [1] and electronics [10]. When it comes to adaptive metamodeling many researchers have tackled the problem of efficient sequential/experimental design [6], adaptive model parameter selection [8], distributed metamodeling [2], and any combination of these [3]. Regarding ANN, a lot of work has gone into applying constructive algorithms (CC, Tiling), Genetic Algorithms (GA), Genetic Programming (GP), Bayesian theory, or other custom algorithms [9] to the difficult problem of selecting the optimal model parameters (network architecture, transfer functions, etc.). A classic example is the EPnet system developed by Yao [9].

```
% Setting up the environment and modeling algorithm
% is really done in the main toolbox control loop,
% driven by the XML configuration file
   algorithm = AnnBatchModeler();
   algorithm.runLoop();
% The runLoop() method calls createBatch()
pop = createBatch(generation,pop):
 if(generation = 0)
  % Create an initial population of mutated default
  % individuals (the default is specified in the XML)
  pop := defaultPop();
  pop := foreach p in pop Mutate(p);
 else
  % Keep the top 20%, re-init the next 40%,
  % and mutate the rest
  [elite, reinited, mutated] := split(pop,20%,40%,40%);
  reinited := foreach p in reinited, reinit(p);
  mutated := foreach p in mutated, mutate(p);
  pop := [elite reinited mutated];
 end
```

```
newNet = mutate(net):
 % Choose the layer to mutate
 layer := randInt(1,2);
 newNet := mutateDimension(net, layer);
 if(newNet.numWeights > k*numSamples)
  % Dont accept architectures that could overfit
  newNet := net;
 end
 if(rand >= 0.7)
  % Reinitialize the new network based on
  % the weights of the old network
  newNet := reinit(newNet,net);
 else
  % Randomly choose new weights
  newNet := reinit(newNet);
 end
 % Randomly change the training function
 newNet.trainFcn := randPick(allowedFcns);
```

Figure 1: The ANN Batch Modeling Algorithm

In short though, all research efforts that we are aware of concentrate on one particular subset of the metamodeling problem in isolation. Instead, we concentrate on developing a set of algorithms/tools that allows one to bring solutions to different subproblems (sequential design, model selection, distributed simulator evaluation) together so they can reinforce each other. Note that we are *not* proposing a universal algorithm that solves every metamodeling problem but rather a flexible research infrastructure wherein different techniques can be easily integrated and compared.

4 Neural Network Modeler

One of the available model types within the M3-Toolbox is the feed-forward multi-layer perceptron (using the standard Matlab Neural Network Toolbox primitives). Most of the standard ANN parameters may be set through the XML configuration file: transfer function, training rule, base architecture, number of epochs, etc. As for the modeling algorithm, used to select the model parameters, a GA-inspired batch algorithm is used with one mutation operator. The algorithm is shown in figure 1.

The problem of searching the ANN parameter space is not a trivial one since it can be expected to be deceptive (cfr. the permutation problem), multi-modal and epistatic. Therefore this evolutionary strategy (ES)-like algorithm is one of the many possibilities and by no means guaranteed to be the best. This algorithm will be updated and improved and other algorithms will be plugged in, next to it, for further study.

As can be seen from the algorithm, the parameters that are varied are: the architecture, the training rule and the initial weights. The weights themselves are not evolved (a fully connected network is assumed) but trained with a fast backprop variant. We admit this is suboptimal since it is known that the architecture and weights are best evolved together [9]. However, due to the added implementation complexity and the increased

running time, this was not done, but will be tackled in the near future when we turn to a full evolutionary based modeling algorithm. In any case, the current mutation operator already produces satisfying results, competitive with multivariate rational modeling.

5 Performance

We now turn to a number of test problems to illustrate our adaptive approach. The ANN modeler will be run on two problems and its performance compared with that of a rational modeler. Note that we could also have included one of the supported SVM modelers (LS-SVM,ϵ-SVM,ν-SVM) in the comparison or increased the number of test problems. However, due to space restrictions, this was not done and such comparisons will be reported on in a future paper. As the first problem we will model Shekel's Foxholes (SF) function, a classic test function from optimization. It is defined as:

$$f(\bar{x}) = -\sum_{j=1}^{30} \frac{1}{||\bar{x}-A(j)||^2+c_j} \text{ with } i = 1..2,\ x_i \in [0, 10]$$

This function is highly non-linear with numerous local minima. As the second test case we take a problem from electronics. A simulation code was used that calculates the complex scattering parameters S_{12} of a Step Discontinuity (SD) in a rectangular waveguide (*see* [4] for a plot). The code takes three real inputs: the signal frequency, the gap height and the gap length. Since the ANN modeler cannot model complex outputs directly, the modulus of S_{12} was modeled. Since it is known from physics that the behavior of a circuit in the frequency domain can be defined as a quotient of two polynomials we should expect the rational modeler to outperform the ANN modeler in this case.

The ANN modeler was configured with a batch size of 10 and each adaptive modeling loop would terminate when (1) the maximum number of batches was reached or (2) there was no improvement in the best model for 3 consecutive batches. The allowed training rules were Levenberg-Marquardt back-propagation (*trainlm*) and Bayesian regularization back-propagation (*trainbr*). If *trainlm* was used the network was trained with early stopping (ratios: 20%:80%). The default hidden layer structure was set to $In - 3 - 3 - Out$ and the default number of epochs to 700. The rational modeler works similarly, except its new models are based on a sliding history of size 30 together with stochastic hill climbing. The parameters varied are the weights of every input parameter, based on which the degree in the (de)nominator is set.

The other toolbox settings were: Latin Hypercube initial experimental design of size 60, sequential design using an adaptive error-based sampler (max 100 new samples every iteration), local (sequential) evaluation, and model scoring using 5-fold cross validation. The toolbox terminates on either of the following conditions: (1) number of samples exceeds 800, (2) a cross validation score on the available samples of 0.1 (SF) or 0.001 (SD) is reached. The performance of the final model was then tested on a testset of 10000 random points with output range [-11.55,-0.97] (SF) or on 20000 random points with range [0.37,1.00] (SD). In both cases the absolute root mean square error (RMS) and maximum absolute error were calculated. The results are shown in table 1 and figure 2 (SF only). The *Final Model* column in table 1 shows the form of

the converged surrogate model. For rational functions, this is the highest degree that appears in the (de)nominator.

Type	*Score*	*#Samples*	*Abs RMS*	*Max Abs*	*Final Model*
ANN	0.18	836	0.09	2.21	2-23-14-1, *trainbr*
Rational	0.31	832	0.31	5.08	$\frac{x^{24}y^{14}}{y^{17}}$
Shekel's Foxholes (SF)					
ANN	0.00088	187	0.00089	0.00723	3-6-5-1, *trainbr*
Rational	0.00052	202	0.00009	0.00070	$\frac{x^9y^5z^{10}}{x^9y^5z^{10}}$
Step Discontinuity (SD)					

Table 1: Test problem results

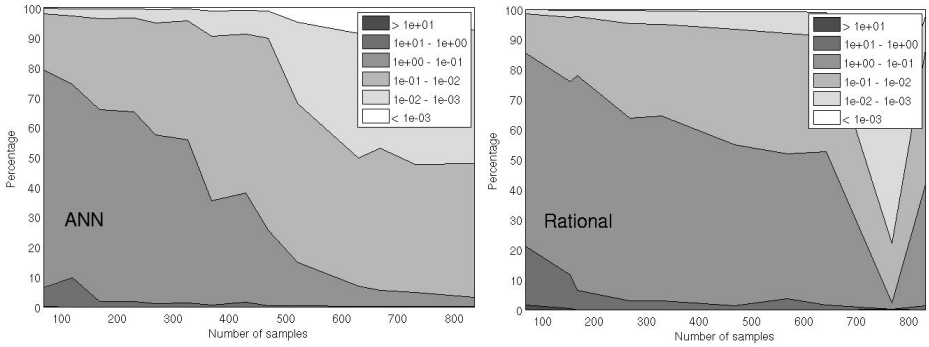

Figure 2: Evolution of the percentage of test samples per error category (SF)

Considering the results, we see that the ANN modeler does quite well. It outperforms the rational modeler on the first problem (SF). The lower score of the rational modeler is caused by the existence of poles in the domain. However, as expected, the rational modeler performs much better on the smoother, second problem (SD). With only a few samples it produces a very accurate metamodel that can easily replace the original simulation code for most practical applications.

6 Evaluation

The results in this paper exemplify our general experience: for highly non linear surfaces ANN tend to do better than rational functions. The reason is that for high-order polynomials the resulting metamodel tends to oscillate near the borders of the domain and poles start to creep in, skewing the errors. Of course, the downside of using ANN is the considerable increase in computation time. For large numbers of data points, many epochs, cross validation, and no early stopping, ANN-based metamodeling results in running times in the order of hours, versus minutes for rational metamodeling.

Finally, note that the goal of this paper was not the comparison *per-se* (plenty of such references exist already, e.g., [5]) but the fact that a comparison is a natural result of tackling the metamodeling problem in an extensible and flexible way. Given the pluggable research platform that the M3-Toolbox provides, it enables almost endless possibilities of comparison between model types, model algorithms and sampling strategies. This information is important for any researcher needing an accurate metamodel at a reasonable computational cost.

References

[1] R. Chaveesuk and A.E. Smith. Economic valuation of capital projects using neural network metamodels. *The Engineering Economist*, 48(1):1–30, 2003.

[2] M. Hakki Eres, Graeme E. Pound, Zhuoan Jiao, Jasmin L. Wason, Fenglian Xu, Andy J. Keane, and Simon J. Cox. Implementation and utilisation of a grid-enabled problem solving environment in matlab. *Future Generation Comp. Syst.*, 21(6):920–929, 2005.

[3] A. Farhang-Mehr and S. Azarm. Bayesian meta-modelling of engineering design simulations: a sequential approach with adaptation to irregularities in the response behaviour. *International Journal for Numerical Methods in Engineering*, 62(15):2104–2126, 2005.

[4] D. Gorissen, K. Crombecq, W. Hendrickx, and T. Dhaene. Grid enabled metamodeling. In *In Proc. of 7th International Meeting on High Performance Computing for Computational Science (VECPAR 2006)*, 2006.

[5] R. Jin, W. Chen, and T.W. Simpson. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23(1):1–13, December 2001.

[6] R. Jin, W. Chen, and A. Sudjianto. On sequential sampling for global metamodeling in engineering design, detc-dac34092. In *ASME Design Automation Conference, Montreal, Canada*, September 2002.

[7] G. Gary Wang and S. Shan. Review of metamodeling techniques in support of engineering design optimization. *ASME Transactions, Journal of Mechanical Design*, page in press, 2006.

[8] R.J. Yang, N. Wang, C. H. Tho, and J. P. Bobinaeu. Metamodeling development for vehicle frontal impact simulation. *Journal of Mechanical Design*, 127(5):1014–1020, September 2005.

[9] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.

[10] Q.J. Zhang, K.C. Gupta, and V.K. Devabhaktuni. Artificial neural networks for RF and microwave design: from theory to practice. *IEEE Trans. Microwave Theory Tech*, 51:1339–1350, March 2003.