2015 International Symposium on Nonlinear Theory and its Applications
NOLTA2015, Kowloon, Hong Kong, China, December 1-4, 2015

Evolutionary Design of Experiments to Analyze Ultra Large Scale Systems

Takeshi Uchitane†

†Advanced Institute for Computational Science, RIKEN
7–1–26 Minatojinma-minami-mnachi Chuo-ku, Kobe, Hyogo 650-0047, Japan
Email: takeshiuchitane@riken.jp

Abstract—Making a design of experiments is very important especially in considering multivariate analyses for ultra large scale systems. It is because that it is too hard to obtain whole or enough analysis data from such systems which have exponentially large parameter space. Thus, a design of experiments which makes it possible to analyze such systems is required. In this paper, a novel framework to dynamically obtain appropriate analysis data sets is proposed. In the framework, an evolutionary algorithm (EA) works so as to optimize analysis results. The results are obtained by analyzing a part of whole data sets. Such sub data set is treated as individuals of EAs. Numerical experiments are performed to make sure that the proposed framework is available to generate appropriate analysis data sets.

1. Introduction

There are many ways to optimize and analyze unknown systems even if the problems are NP–hard. In a context of optimization, for example, evolutionary computations get much attentions because the algorithms are able to minimize or maximize the system outputs and we usually can obtain enough better solutions without evaluating whole system outputs. On the other hand, in a context of analysis, we still have many trial and errors to obtain an insight about the systems. Via analyzing systems, we usually try to find three things, how many factors and what factors are in the system and how the factors affect the system outputs. To achieve to gets such insights, we have to obtain enough data sets of system inputs and outputs. And the number of data sets easily get larger as the number of system model parameters are increasing. Nowadays, we believe that such data is called big data even if the data sets are obtained by numerical experiments using computers.

It may be possible to analyze ultra large systems with fewer number of system input–output sets only when a degree of freedom of analysis models is less than or equal to the number of factors n. The ideas have been studied in space modeling. For example, in finding brain areas predictive about brain states based on fMRI data, billions of neural activity are described by several factors[2]. Thus, it is required to estimate the number of factors and what factors are from such big data.

The goal of this study is to develop a design of experiments to estimate factors of systems with fewer number of analysis data sets obtained by executing numerical simulations about the systems. A typical design of experiments, for example, is analysis of variance to find factors of systems[1]. Applying analysis of variance to data sets, we can estimate factors though candidates of factors are given before analyzing the data sets. Additionally, using orthogonal arrays makes it is possible to reduce the number of experiments $O(n)$ from $O(n^n)$. However, n as the number of candidates of factors is still large in considering ultra large scale systems .

In this paper, an analysis that simulation results are classified into two classes. In the analysis, data sets are described by a system input as real numbers vector and a system output as classes. The number of factors is given before analyzing but what factors are is going to be estimated. The patten classifier as the analysis learns a training data sets as system outputs. Then, a novel framework as a design of experiments works to obtain appropriate sub data sets. In the framework, evolutionary computation techniques are applied to dynamically obtain such sub data sets. So, the proposed framework is called *Evolutionary Design of Experiments* in sort *EDoEs* here. The availability of proposed framework is discussed via several numerical experiments.

2. Definition of Evolutionary Design of Experiments

In this section, a definition of *EDoEs* is described. In *EDoEs*, there are three components, *Simulator*, *Analyzer* and *Evolutionary algorithm* (in short *EA*). *Simulator* is given but is treated as an non–linear function. There are several kinds of *Analyzer* as multi–variable analyses, e.g. regression analysis, analysis of variance, machine learning, data mining and so on. *Analyzer* has common features to estimate the number of factors and estimate how factors affect the perspectives. *EA* optimizes the result of *Analyzer* for the output data sets of *Simulator*. For example, in considering neural network (in short *NN*) as an *Analyzer* learning classification of simulation outputs, a fitness function is defined as the rate of classification.

There are two goals of *EDoEs*, one is to obtain fewer appropriate data sets and another is to optimize results of analyses. In other words, enough data sets can be obtained but whole data sets can not be obtained. To obtain such data set, there are following steps in *EDoEs*. In the first step, initial simulation parameters are generated at random. In the second step, *Simulator* makes the simulation outputs. In the third step, the result of an *Analyzer* is obtained. In

the forth step, a new set of simulation parameters are generated by *EA*. After the forth step, steps from the second to the forth are repeated by conditions about the result of the *Analyzer* are satisfied.

3. Numerical Experiment

In this section, numerical experiments are shown to estimate availability of *EDoEs*. Results of *NN* as an *Analyzer* using data sets obtained by *EDoEs* are compared with data sets which are generated randomly or uniformly. In the following paragraphs, *Simulator*, *Analyzer* and *EA* in numerical experiments are defined.

3.1. Definition of *Simulator*

A function $F_{\mathbf{S}}(\cdot)$ as *Simulation* is defined by

$$F_{\mathbf{S}}(\mathbf{x}) = \begin{cases} label1 & (\mathbf{x} \in \mathbf{S}) \\ label2 & (\mathbf{x} \notin \mathbf{S}) \end{cases} \quad (1)$$

where $\mathbf{x}$ is a real vector $([x_1, x_2, \ldots, x_d, \ldots, x_D]^T)$ and $\mathbf{S}$ is a sub set of $\mathbb{R}^D$. Then, $\mathbf{S}$ is defined in table 1.

Table 1: A list of *Simulator*. Names of *Simulator* and region of S are shown. a, b, c and d are constant values.

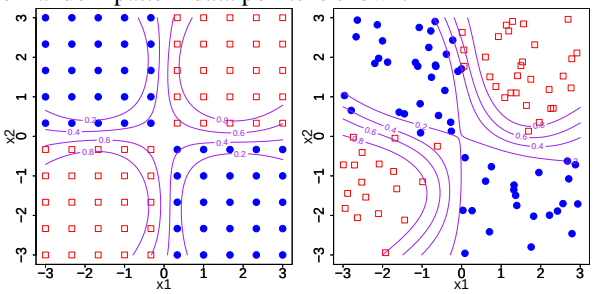
Test name	$\mathbf{S}$
XOR	$XOR(\sum x_{2d-1} > 0, \sum x_{2d} > 0)$
SingleRectangle	$a < x_d < b$
DoubleRectangle	$OR(a < x_d < b, c < x'_d < d)$ s.t. $b < c$
HyperCone	$\sum a_d x_d < b$ and $x_d > c$

3.2. Definition of *NN* as an *Analyzer*

An usage to classify data set into several groups by using *NN* is popular in the field of machine learning. *NN* can lean patterns from a training data set and classify new inputs. Applying a state of the art *NN* in *EDoEs* may be successful, but studies to improve *EDoEs* is beyond the goal of this paper which makes sure the availability of *EDoEs*. Therefore, a single-hidden-layer *NN* introduced by Venables et. al.[3] is applied as *Analyzer* here.

An example of patten classification using *NN* for *XOR*. Here, the number of training data is one hundred or more but the number of factors n is equal to two. An input $x_1 = -2$ and $x_2 = 2$ is classified to *label*2. A comparison training data set is obtained at random from uniform distribution ($\mathbb{U}[-3, 3]$) and another training data set is distributed in a uniform grid pattern. (See figure 1.) Nodes in *NN* is defined as a sigmoid function and *NN* outputs a scalar value $0 < y_i < 1$ with an input $\mathbf{x}$. *label*1 is associated with output y is larger than 0.5 and *label*2 is associated with y is less than 0.5. Both the number of nodes in a middle hidden layer (N_i) and network weights are tuned by R package "caret"[4]. It means that the number of N_i may be different in each training data set.

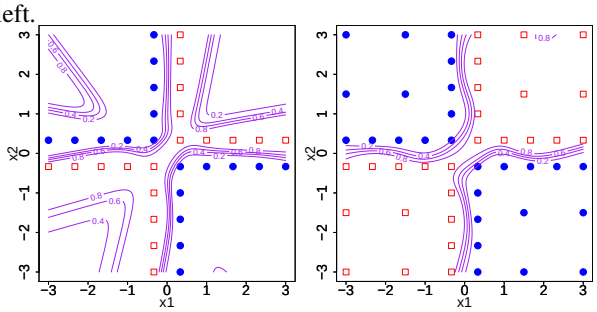
Figure 1: Examples of pattern classification learning for *XOR* test function. *label*1 is described as a mark of rectangle and *label*2 is described as a mark of circle. Level curves of *NN* as a probability density function are drown. In the left figure, a pattern classifier for grid pattern data points is shown and in the right figure, a pattern classifier for random pattern data points is shown.



3.3. Definition of *EA*

EA works so as to optimize the results of *Analyzer* in the proposed framework. It means that *EA* finds candidates of training data set so as *NN* can learn appropriate patterns. Here it is assumed that *EA* keeps balance between finding borders between labels and covering parameter space widely. Actually the result of *NN* shown in the left of figure 2 seems better than the result via randomly sampled training data sets. But *NN* can not classify correctly in some areas far from the border. It is because that training data is only concentrated nearby the border. On the other hand, the result shown in the right of figure 2 seems not worse than the grid pattern even if the number of training data is less than the grid pattern. Thus, *EA* should find training data set not only around the border but also covering result space widely.

Figure 2: Results of pattern classification for *XOR* test function with fewer data points. The left figure shows the results of *NN* with data points nearby borders. And the right figure shows that results of *NN* with data points not only nearby borders and also a few additional points far from border. The result in the right figure is better than the left.



In considering to find new candidates of training data, it is required to mainly search around borders and also to

obtain widely distributed data points for each label. Thus, from the context of genetic algorithm (GA), methods of *crossover* with *selection* and *mutation* are employed to generate a new set of candidates from evaluated training data sets. The *crossover* tries to find new training data so as to obtain data which are placed near by borders. The *selection* tries to select better candidates from found training data sets via *crossover*. The *mutation* tries to find new training data so as to obtain data which are placed far from borders.

To generate training data points around borders (it calls *border points*), a data sampling method inspired from crossover in GA is proposed. In considering *border points* are distributed between a *label1* point and a *label2* point. So, a new candidate of *border points* is generated by a following equation.

$$|x_{new} - x_{label1}| = |x_{new} - x_{label2}| \tag{2}$$

where x_{new} is a new candidate of *border points*, x_{label1} is a randomly selected point from training data points which belong to S_{test} and x_{label2} is a randomly selected point from training data points which do not belong to S_{test}.

To obtain better *border points*, the *selection* inspired from natural selection in genetic algorithm is applied. In genetic algorithm, each individual is evaluated by an evaluation function. So, an evaluation function to find such training points is defined as following equations.

$$\text{Maximize } fitness \tag{3}$$
$$fitness = |0.5 - F_{NN}(x)|, (0 < F_{NN}(x) < 1)$$

where x is a new data point and $F_{NN}(x)$ is estimated probability density function as *NN*. In the evaluation function, data points which are placed in borders estimated by *NN* can get larger $fitness$. Generated *border points* are selected based on *fitness*. Here, a selection randomly selected from top 50% generated points is applied.

To generate training data points far from*border points*, a sampling method inspired from mutation in GA is proposed. a new candidate of training data is randomly sampled from $\mathbb{U}[-3, 3]$.

Processes to obtain a training data set by using *EA* is shown in following steps.

- Initial training data points are generated.
- *NN* learns whole generated data points.
- Additional data sets are found by *EA*.
  - New candidates of data sets are generated via *crossover* and *mutation*.
  - Generated data sets are evaluated by the evaluation function.
  - Output N data points randomly selected from top of 50% points based on *fitness*.
- The additional points generated by *EA* are added into training data set.
- The above three steps are repeated until termination.

4. Numerical Experiments

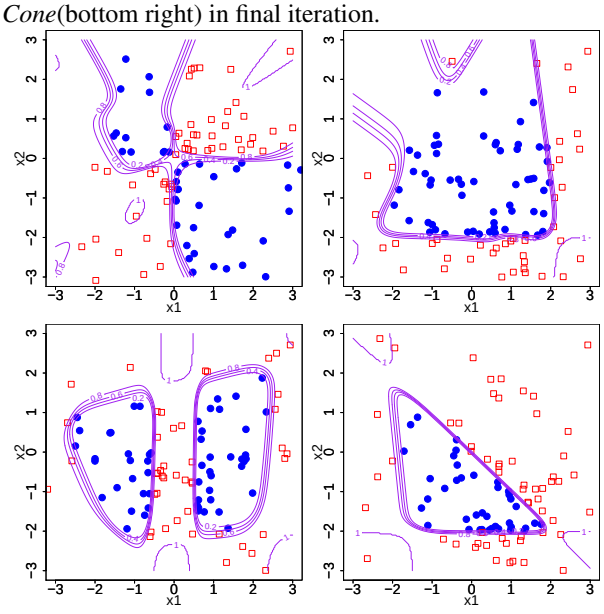
In this section, numerical experiments are shown in order to make suer that the proposed framework is available to generate appropriate training data sets with fewer number of data points. Common settings for numerical experiments are defined as the following table.

Table 2: A list of *Simulator*: names of *Simulator* and regions S are shown.

settings	2 dim.
# of final training data points (N_{max})	100
# of *EA* iteration	10
# of generated data points by *EA* (N)	10
crossover	sampling *border* points
crossover rate (N_c)	8 over N
mutation	sampling from $\mathbb{U}$[-3, 3]
mutation rate (N_m)	2 over N

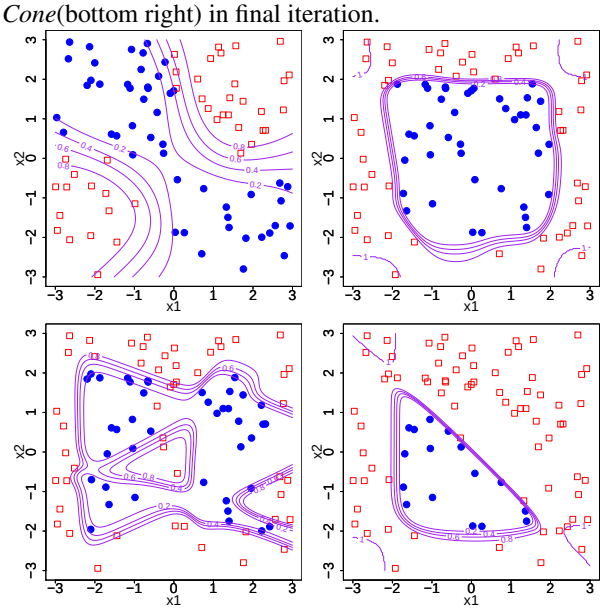
Fig. 3 shows examples of the data points in final iteration each test, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right) respectively. It seems that training data concentrates around the border points in each S. Fig. 4 shows examples of pattern classification from randomly sampled data points for each test, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right) respectively. Fig.5 shows iteration–

Figure 3: Results of pattern classification via proposed framework for each test, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right) in final iteration.



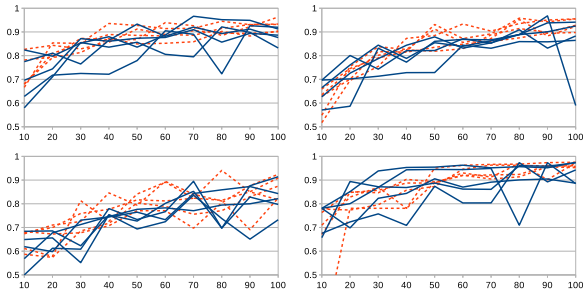
rate plots of *NN* for each test, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right). Solid five lines represent transition of

Figure 4: Results of pattern classification from randomly sampled training data, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right) in final iteration.



the rates of answered correctly in proposed method. On the other hand, dashed five lines represent transition of the rates of answered correctly in sampling randomly from $\mathbb{U}$[-3, 3]. Rates of *NN* answered correctly increase as the number of training get large.

Figure 5: Plots of iteration–rate of *NN* for each test, *XOR*(top left), *SingleRectangle*(top right), *DoubleRectangle*(bottom left) and *HyperCone*(bottom right) in final iteration. Five solid lines represent the rate transitions with training data obtained by proposed method and five dashed lines represents the rate transition with randomly sampled data.



5. Discussion

It seems that training data obtained by *EDoEs* concentrates around the border between labels in each test. So *EDoEs* can keep balance between finding *border points* and finding widely spreading points. This may implies that it is possible to obtain a enough number of *boder points* via *EDoEs* in high dimensional problem.

The rates of answered correctly via *EDoEs* are not different from via randomly sampled data. This is because that the number of dimension of parameter space is only two. It seems that we may find differences in high dimension parameter spaces.

6. Conclusion

In this paper, a novel framework to dynamically obtain an appropriate analysis data sets via numerical simulations. The proposed framework is defined by three components *Simulator* as a no linear function, *Analyzer* as a multivariate analysis and *EA* as design of experiments to optimize the result of the *Analyzer*. In order to make sure the availability of the proposed framework, numerical experiments are performed. From the results of experiments, it is shown that it is possible to obtain some appropriate data sets by using the proposed framework.

Making sure that availability of the proposed framework for high dimensional *Simulator* will be expected. Moreover The *NN* which is used in the experiments is a general neural network. Applying online learning[5] as an *Analyzer* and effective training data generation in *EA* are under consideration to reduce the calculation cost to obtain training data set.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15K17502.

References

[1] W. G. Cochran and G. M. Cox, “Experimental Design”, 2nd–ed. John Wiley & Sons, 1950.

[2] M.K. Carroll, G.A.Cecchi, I. Rish, R. Garg, A.R. Rao. “Prediction and Interpretation of Distributed Neural Activity with Sparse Models”, Neuroimage 44(1):112-22, 2009.

[3] W. N. Venables and B. D. Ripley, “Modern Applied Statistics with S”, 4th ed. Springer, 2002.

[4] M. Kuhn, “caret: Classification and Regression Training”,`https://cran.r-project.org/web/packages/caret/index.html`

[5] F. Shen and O. Hasegawa, “An Incremental Network for On-line Unsupervised Classification and Topology Learning”, Neural Networks, Vol. 19, No. 1, pp. 90–106, 2006.