Use of RBM for Identifying Linkage Structures of Genetic Algorithms

Hisashi Handa

School of Science and Engineering
Kindai University
Kowakae 3-4-1, Higashi-Osaka, Osaka 577–8502, Japan
Email: handa@info.kindai.ac.jp

***Abstract*—Linkage Identification is a quite important in Genetic Algorithm studies. If we found linkage structures, we can improve coding methods, genetic operations. The Restricted Boltzmann Machine (RBM) is adopted to capture linkage structures in this study. The learning data of the RBM consist of data generated by referring to the nonlinearity check of the Linkage Identification by non-Linearity Check. The activation values of the neurons in the hidden layer in the RBM and corresponding learning data are visualized. We can easily find out linkage structures by using the resultant images.**

Keywords–Genetic Algorithms; Linkage Identification; Restricted Boltzmann Machines.

I. INTRODUCTION

Linkage Identification plays crucial roles in Evolutionary Computation. The linkage means that a certain tuple of genes appears with a higher rate more than the expected value of a tuple of independent genes [1]. Such genes with higher linkages are supposed to be functionally dependent. Based upon the Building Block Hypothesis [2], several building blocks which have higher linkages are juxtapositionally searched. They are combined with each other by using crossovers implicitly. Therefore, the GA community has been studying this area, since a wrong problem encoding method with strong linkage causes of performance deteriorations [3]. We can elaborate genetic operators if we found the linkage structure in advance. Although Estimation of Distribution Algorithms (EDA) are a systematic approach to solving problems taking into account for problem structures, some EDA, such as the Distribution Estimation Using Markov network (DEUM) still need linkage structures in order to design Markov Networks [4][5].

In this paper, we use RBM to capture the linkage structures. That is, we do not pursue to develop a new mechanism to detect linkage structures. We can detect linkage structure by seeing the visualized image as a consequence of the learning of the RBM.

The organization of the remainder of the paper is as follows: Section II briefly summarizes related works. Section III introduces Linkage Identification by Nonlinearity Check (LINC). We employ the non-linearity check in the LINC to detect the non-linearity in the proposed method. Section IV explains the RBM. Section V describes the proposed method. Preliminary experiments on several benchmark problems are examined in Section VI. Section VII concludes this paper.

II. RELATED WORKS

One aspect of linkage identification problems is that they can be regarded as a permutation problem [3]. In genetic algorithms we usually use bit string representation. Therefore, in order to address linkage identification problems, we can solve for an ordering problem of individual representation. The effectiveness of the inversion operator is investigated by Bagley [6], where the inversion operator reverses the order of a partial sub-string in an individual. The idealized reordering operator introduced by Goldberg and Bridges [7] can address the linkage learning problems. Ying-ping showed the performance of the genetic algorithms with the idealized reordering operator can be improved by using tournament selection [8].

Conventional linkage identification studies tackle to devise the mechanism to detect linkages: LINC by Munetomo *et al.* uses non-linearity check among two loci to find out the linkage [9]. His group has been extended to cope with more realistic problems, such as Linkage Identification with Epistasis Measure considering monotonicity (LIEM), and Linkage Identification by non Monotonicity Detection (LIMD) [10][11].

Our main contribution of this paper is to tackle to address of the linkage identification problems by using representational learning algorithms, i.e., the RBM.

III. LINKAGE IDENTIFICATION BY NON-LINEARITY CHECK

LINC by Munetomo *et al.* focused on the non-linearity among two loci. This paper adopts this nonlinearlity check method in order to generate learning data of the RBM.

Suppose that s indicates an individual whose string length is L. In order to investigate the linkage structures, s is randomly sampled at first. Base upon the individual s, nonlinearity is checked as followings: Now, $\Delta f_i(s)$ stands for the fitness difference if the i^{th} locus is changed:

$$\Delta f_i(s) = f(..\bar{s}_i....) - f(..s_i....) \quad (1)$$
$$\Delta f_j(s) = f(.....\bar{s}_j.) - f(.....s_j.) \quad (2)$$
$$\Delta f_{ij}(s) = f(..\bar{s}_i..\bar{s}_j.) - f(..s_i..s_j.), \quad (3)$$

where $\Delta f_{ij}(s)$ means the fitness difference if the i^{th} and j^{th} loci are simultaneously changed.

Equation (4) is used if the non-linearity exists among the i^{th} and j^{th} loci:

$$|\Delta f_{ij}(s) - (\Delta f_i(s) + \Delta f_j(s))| > e, \quad (4)$$

where e is a small positive number.

 ISBN: 978-1-61208-461-9

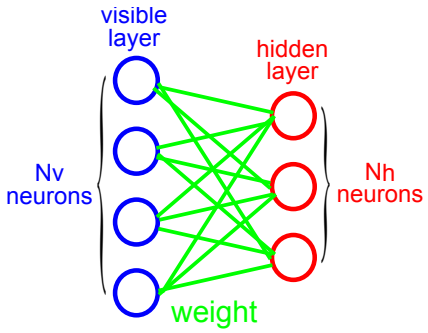

Figure 1. Depiction of Restricted Boltzmann Machine

IV. Restricted Boltzmann Machines

RBM is a two-layered network as shown in Figure 1: The visible layer v and the hidden layer h. Each neuron in the visible layer v is fully connected with all the neuron in the hidden layer h. Note that there are no connections among neurons in the visible layer and among neurons in the hidden layer.

Energy function $E(\mathbf{v}, \mathbf{h})$ can be defined as follows:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i}^{N_h}\sum_{j}^{N_v} w_{ij} h_i v_j - \sum_{j}^{N_v} b_j v_j - \sum_{i}^{N_h} c_i h_i, \quad (5)$$

where N_h and N_v denotes the number of neurons in the hidden layer h and in the visible layer v, respectively. h_i indicates i^{th} neuron in the hidden layer. v_j is j^{th} neuron in the visible layer. w_{ij}, b_j, and c_i are learning parameters for the RBM: weight values, bias values for the visible layer, and bias values for hidden layer. Parameters, such as w_{ij}, b_j, and c_i, are learned by using Constrictive Divergence method.

The joint probability $P(\mathbf{v}, \mathbf{h})$ is represented by using this energy function $E(\mathbf{v}, \mathbf{h})$ in equation (6):

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v}, \mathbf{h})), \quad (6)$$

where Z is a normalized factor.

For input $\mathbf{v}$, the activation values of the neurons in the hidden layer is calculated as follows:

$$p(h_i = 1|\mathbf{v}) = \sigma(c_i + \sum_{j=1}^{N_v} w_i j v_j), \quad (7)$$

where σ means Sigmoid function.

V. Proposed Method

A. Overview

This paper uses the RBM to identify the linkages in problems. First, the non-linearity check in (4) is used to detect loci which may have linkages. Such loci information is transformed into learning data. These learning data are learned by using the RBM. After the learning of the RBM, the learned RBM examines the learning data again in order to investigate the activated values for the learning data. The averaged vector of the learning data, such that a certain neuron in the hidden layer is activated, is calculated. An image is generated by using a set of the averaged vectors for all the neurons in the hidden layer. From the image, we can find out if the linkage exists.

1: **repeat**
2: An individual s is randomly generated.
3: **for** each pair $(i, j) \in A$ **do**
4: **if** rand() $< p$ **then**
5: The pair (i, j) is examined the non-linearity check in (4).
6: **if** non-linearity is found for the pair (i, j) **then**
7: A learning data based on the pair (i, j) is appended.
8: **end if**
9: **end if**
10: **end for**
11: **until** The number of learning data reaches to the predefined number

Figure 2. Procedure of generating learning data

```
0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0
0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0
0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0
```

Figure 3. Learning data generated by the procedure in Figure 2

B. Generation of Learning Data

We assume that in order to solve a given problem to be optimized, i.e., fitness function, a binary encoding method with L bits is used. That is, individuals s are represented by a L bit string. Figure 2 summarizes the procedure of generating learning data, where A denotes a set of all the combinations of two numbers picked from 1 to L without repetition. The "rand()" at line 4 in the procedure is supposed to be random function which returns a random number in $[0, 1]$ over the uniform distribution. A parameter p at line 4 is set to be 0.05 in this study. The number of learning data at line 11 in the procedure is set to be 100,000.

The generation of learning data at line 7 in Figure 2 is described as follows: First, an L bit-string is generated, where all the data in the string is set to be 0. Second, flip the i^{th} and j^{th} data to 1. Figure 3 shows an example of learning data generated by the procedure in Figure 2. Each row denotes each learning data so that this problem seems to be the one of 27-bits length. At the first row, the second and the third loci is set to be 1. Hence, this learning data implies these loci may have a linkage at these loci. This example only shows 10 data while the number of learning data in this study is set to be 100,000.

These learning data are applied to the RBM. The number of neurons in the visible layer N_v is the same as the string length L. The number of neurons in the hidden layer N_h is set to be around 90 % of N_v. These numbers are varied for problem instances; the exact number of N_v and N_h is explained in Section VI.

 ISBN: 978-1-61208-461-9

C. Generation of Averaged Vectors

As depicted in Figure 4, after the learning of the RBM, the learned RBM examines the learning data again in order to investigate the activated values of the neurons in the hidden layer. The input pattern, i.e., each of learning data and the corresponding activated values, is stored.

Suppose that u_i denotes a set of learning data, such that the i^{th} neuron in the hidden layer is activated. In this paper, the hidden neuron is activated if the activated value is greater than 0.5. The average vector $\mathbf{x}_i^*$ is averaged over the entire learning data in the set u_i. Figure 4 shows an example, where the averaged vector is calculated when the second neuron in the hidden layer is activated. The averaged vector $\mathbf{x}_i^*$ is a real-valued vector with the L dimensions. The range of the elements of the averaged vector $\mathbf{x}_i^*$ is [0, 1], since each learning data is a binary vector.

D. Visualization of the Averaged Vector

A binary vector $\mathbf{b}_i = (b_i^1, b_i^2, \ldots, b_i^L)$ is generated from each of the averaged vector $\mathbf{x}_i^* = (x_i^{*,1}, x_i^{*,2}, \ldots, x_i^{*,L})$, where $i = (1, 2, \ldots, N_h)$: Each b_i^j is set to be 1 if $x_i^{*,j} > 0.5$. Otherwise, b_i^j is set to be 0.

These binary vectors $\mathbf{b}_i$ are only used for sorting the averaged vectors $\mathbf{x}_i^*$. That is, these binary vectors are regarded as binary numbers so that these binary numbers are used as a key for the sort. Based upon the key, the $\mathbf{x}_i^*$ is sorted. The sorted $\mathbf{x}_i^*$ is used to generate an image, such that the ping or red color at a certain pixel indicates there may be linkage at corresponding variables.

Figure 5 shows an example of generated image. Each averaged vector is delineated horizontally. Each element of the averaged vector $\mathbf{x}_i^*$ is assigned 10x10 pixels. Hence, the image has the width $L \times 10$, and the height $N_h \times 10$.

As in Figure 5, white color denotes that the corresponding element of the averaged value is 0, which means it has no linkage. Meanwhile, red color means the corresponding element has linkage with the other colored element in the same row. Pink color indicates marginal: strong pink denotes that it tends to have linkage while light pink denotes that it may have linkage.

VI. Experiments

A. Fitness Functions

- **n-trap function** has n-bit linkages. Suppose that The n-trap function is composed of m sub-functions f_t, where $m = L/n$:

$$f_t(\mathbf{s}, k) = \begin{cases} n & \text{if } n_k = n \\ n - 1 - n_k & \text{Otherwise,} \end{cases} \quad (8)$$

  where n_k indicates the number of one's in the k^{th} sub-function.
  Fitness function f_{trap} can be defined as follows:

$$f_{\text{trap}}(\text{s}) = \sum_k^m f_t(\text{s}, k) \quad (9)$$

- **6-trap bi-polar function** is similar to the n-trap function where $n = 6$. The sub-function f_b of the 6-trap bi-polar function, however, has two peaks.

$$f_b(\mathbf{s}, k) = \begin{cases} 3 & \text{if abs}(3 - n_k) = 0 \\ 2 - \text{abs}(3 - n_k) & \text{Otherwise,} \end{cases} \quad (10)$$

  where n_k is the same meaning as in the n-trap function.
  Fitness function f_{6trapBP} can be defined as follows:

$$f_{\text{6trapBP}}(\text{s}) = \sum_k^{L/6} f_s(\text{s}, k) \quad (11)$$

- **hierarchical trap function** can be recursively defined: there are a large number of 3-trap-like function, which are connected as in a tree. For explanation, we assume that $L = 27$. We have nine 3-trap-like function f_h^1 at the bottom level as follows:

$$f_h(\mathbf{s}, k) = \begin{cases} 3 & \text{if } n_k = 3 \\ 3 - n_k & \text{Otherwise,} \end{cases} \quad (12)$$

  If $n_k = 3$ or $n_k = 0$, such function outputs 1 or 0 to the upper level, respectively. Otherwise, the function output nil.
  In the second level, i.e., the upper level of the bottom level. We have three 3-trap-like function. The outputs from the bottom level are used for fitness calculations. Therefore, each 3-trap-like function at the second level is connected to three 3-trap-like functions at the bottom level. The 3-trap-like functions output nil if there is/are (an) output(s) nil from at least one 3-trap-like function at the lower level. Otherwise, 3-trap-like function f_h in (12) is used.
  At the top level, there is a 3-trap function. As in the 3-trap-like function at the second level, this function at the top level uses the output from the functions at the lower level, i.e., the second level. This function at the top level also output nil if at least one 3-trap-like function(s) output(s) nil.
  The fitness of the hierarchical trap functions is calculated by summing all the outputs of the 3-trap-like function at the top level and the one of 3-trap-like functions at the other level. Note that nil is regarded as 0 for this calculation.

TABLE I. FITNESS FUNCTIONS AND THE NUMBER OF NEURONS IN THE VISIBLE LAYER N_v AND THE HIDDEN LAYER N_h

fitness function	$L(= N_v)$	N_h
5-trap	50	40
six-bipolar	48	40
hierarchical trap	27	24
hierarchical trap	81	72

B. Experimental Settings

As described in Table I, we examined several fitness functions explained the previous subsection. Except for the hierarchical trap function, all the functions are decomposable functions. Table I also summarizes the number of neurons in the visible layer N_v and the hidden layer N_h of the RBM.

 ISBN: 978-1-61208-461-9

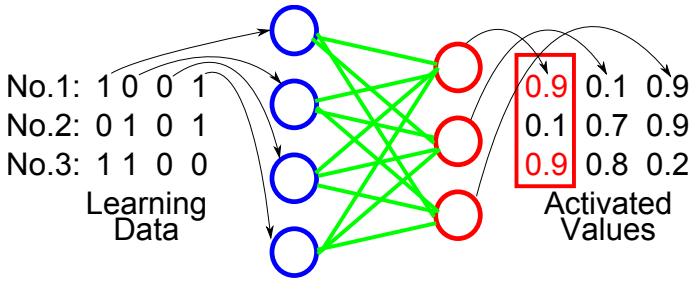

Figure 4. Learning data generated by the procedure in Figure 2

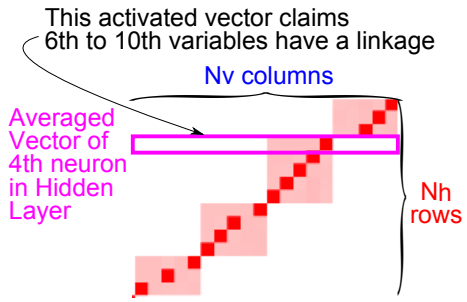

Figure 5. Example of visualization

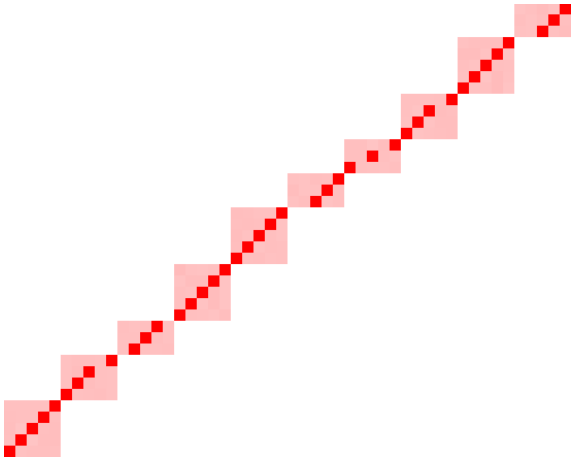

Figure 6. Experimental Results for 5-trap function

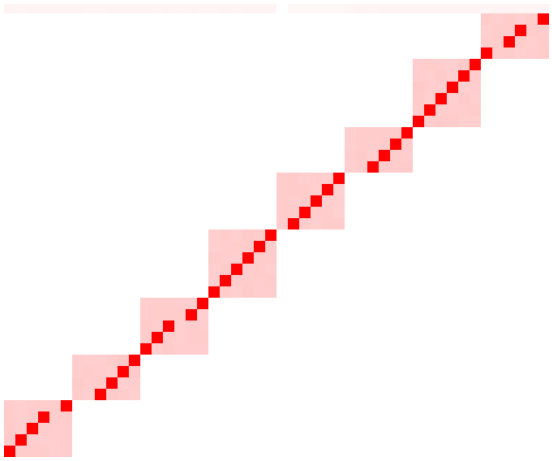

Figure 7. Experimental results for 6-trap bipolar function

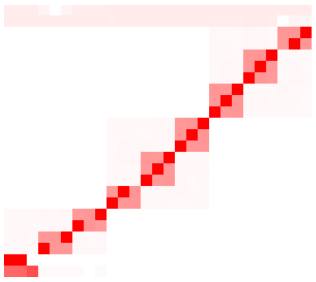

Figure 8. Experimental results of 27-bit Hierarchical Trap Function

C. Experimental Results of Decomposable Functions

Figures 6 and 7 show the experimental results for the 5-trap function and the 6-trap bipolar function, respectively. There are ten and eight linkage clusters in these functions. Even though the non-linearity check examined in this paper concerns with only two bits, the proposed method can capture such linkage structures well.

The first averaged vector in Figure 7 depicted lighter pink for all the variables. Such lighter pink patterns are often appeared.

D. Experimental Results of Hierarchical Trap Function

Figures 8 and 9 show the experimental results of the non-decomposable function, i.e., the Hierarchical Trap function. We examined 27-bit Hierarchical Trap Function — three level problem —, which is explained in subsection VI-A. Moreover, 81-bit Hierarchical Trap Function — four level problem — is also examined. These figures show that the proposed method can find out linkage structures for the bottom level and the second level. For the second level, we can see that the lighter pink areas are for each tuple of corresponding nine variables. However, the third level and the fourth level (for 81-bit problem) cannot extract linkage structure at all. The reason for this is that it is unable to capture such dependence structure by only two bits which are examined in the non-linearity check in this paper. Therefore, we think that non-linearity check with the higher order is needed.

 ISBN: 978-1-61208-461-9

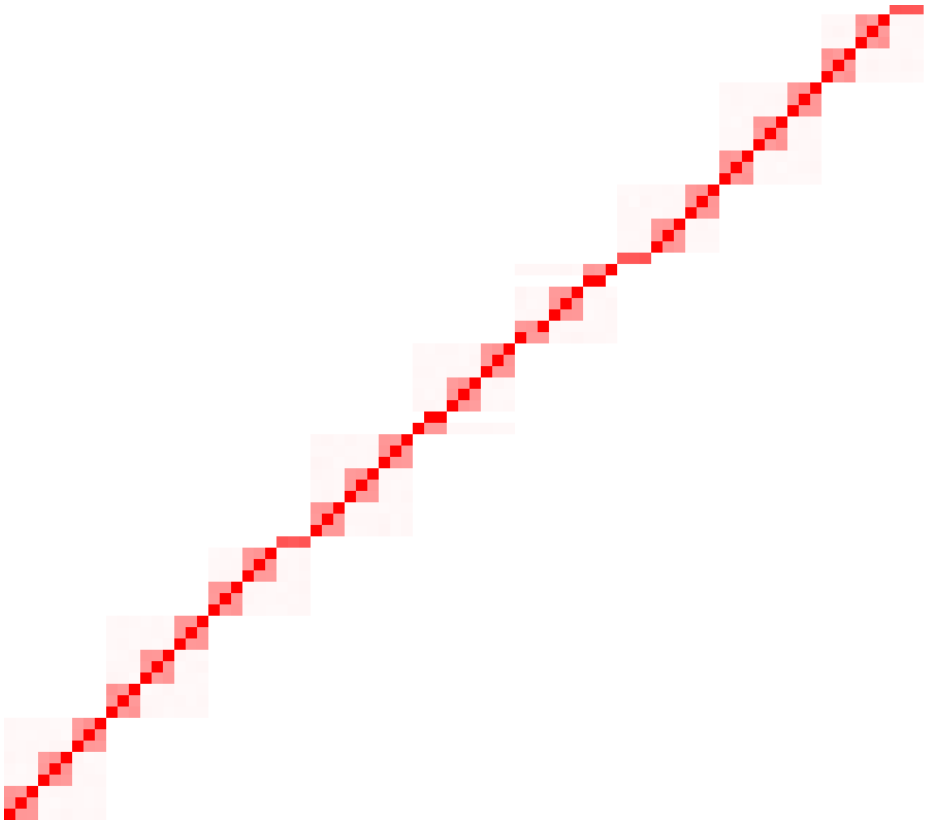

Figure 9. Experimental results of 81-bit Hierarchical Trap Function

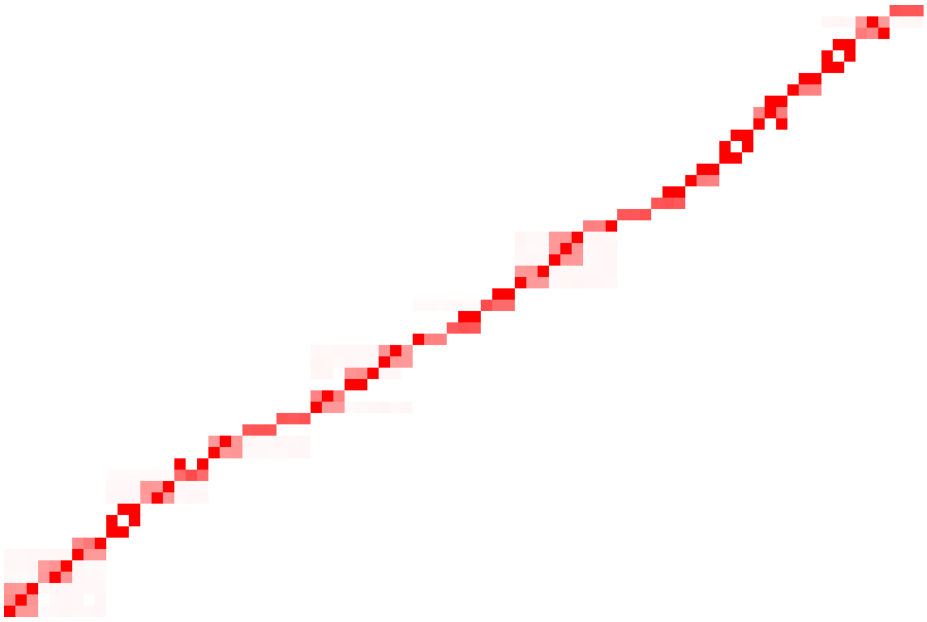

Figure 10. Experimental results of 81-bit Hierarchical Trap function by the Deep Boltzmann Machine

E. Comparison with "Deep" Structures

Recently, the RBM is used in Deep Learning, which is a part of Deep Belief Net., or Deep Boltzmann Machine. We examine the effect of deep structure in neural networks for linkage detection. In Figure 10, the experimental results of the DBM are shown. We employed 81-bit Hierarchical Trap function for this comparison. The network structure of the DBM, i.e., the number of neurons in each layer, is 81, 72, 63, and 54. For comparison, we constitute the RBM such that the number of neurons in the visible layer and the hidden layer are 81 and 54, respectively. The experimental results

 ISBN: 978-1-61208-461-9

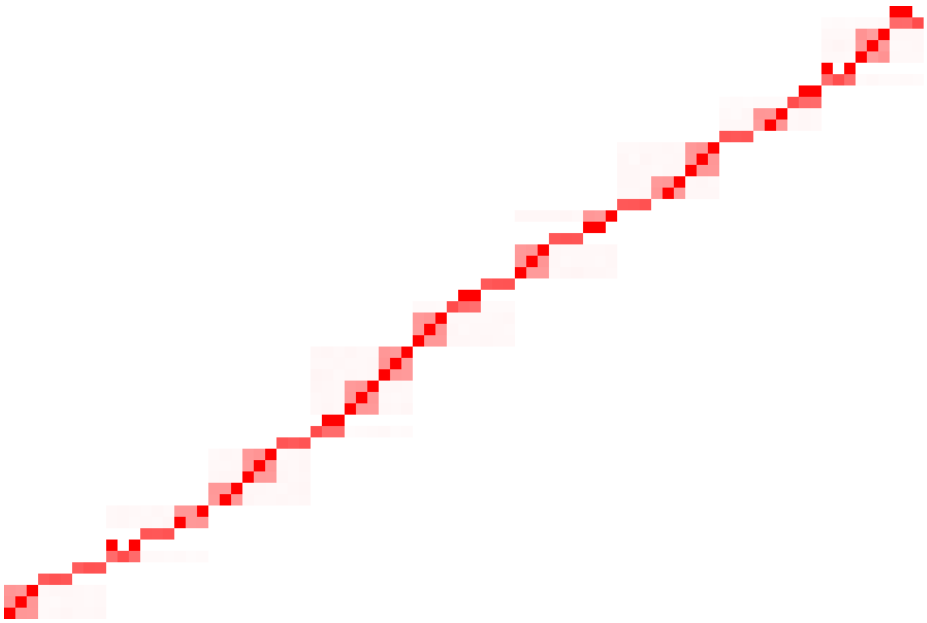

Figure 11. Experimental results of 81-bit Hierarchical Trap function by a single RBM with 54 neurons in the hidden layer

of the RBM are shown in Figure 11. It seems that there are no significant differences between these figures. Even if deep structures are employed, we cannot detect linkage structures in the third and fourth levels. We might need to examine more various network structures for the DBM. However, we think that the input for the RBM and the DBM are modified in order to capture linkage structure, i.e., non-linearity check so that such higher dependency are disregarded at the learning data generation.

VII. Conclusion

In this study, we employed the RBM to detect linkage structures in fitness function which are used in Genetic Algorithms. In order to constitute the learning data for the RBM, non-linearity check used in LINC is used. We also introduced a visualization method by using sorted averaged vectors. We can easily find out by using the resultant images if the linkage structures exist.

Future works are summarized as follows: We extend non-linearity check mechanism to cope with a higher order of linkage detection. We think that more bits should be simultaneously checked. On the other hand, we would like to use the deep learning in this study. For this purpose, the representation of learning data must be considered.

Acknowledgment

This work was partially supported by the Grant-in-Aid for Scientific Research (C) and the Grant-in-Aid for Young Scientists (B) of MEXT, Japan (26330291, 23700267)

References

[1] D. R. Newman, “The Use of Linkage Learning in Genetic Algorithms,” http://www.ecs.soton.ac.uk/ drn05r/ug/irp/, Last Update: September 2006.

[2] D. E. Goldberg, “Genetic Algorithms in Search, Optimization, and Machine Learning,” Addison-Wesley, 1989.

[3] C. Ying-ping, Y. Tian-li, S. Kumara, and D. E. Goleberg, “A Survey of Linkage Learning Techniques in Genetic and Evolutionary Algorithms,” Technical Report IlliGAL Report, 2007.

[4] P. Larranaga and J. A. Lozano (Eds.), “Estimation of distribution algorithms: A new tool for evolutionary computation,” Kluwer Academic Publishers, 2002.

[5] S. K. Shakya, J. A. W. McCall, and D. F. Brown, “Updating the probability vector using MRF technique for a Univariate EDA,” Proceedings of the Second Starting AI Researchers ’ Symposium, volume 109 of Frontiers in artificial Intelligence and Applications, 2004, pp. 15-25.

[6] J. D. Bagley, “The behavior of adaptive systems which employ genetic and correlation algorithms,” Ph.D. dissertation, University of Michigan, Ann Arbor, MI, 1967.

[7] D. E. Goldberg and C. L. Bridges, “An analysis of a reordering operator on a GA-hard problem,” Biological Cybernetics, vol. 62, 1990, pp. 397-405.

[8] C. Ying-ping and D. E. Goleberg, “An analysis of a reordering operator with tournament selection on a GA-hard problem,” Proceedings of Genetic and Evolutionary Computation Conference 2003 (GECCO-2003), 2003, pp. 825-836.

[9] M. Munetomo and D. E. Goldberg, “Designing a genetic algorithm using the linkage identification by nonlinearity check.” Technical Report IlliGAL Report No.98014, University of Illinois at Urbana-Champaign, 1998.

[10] M. Munetomo and D. E. Goldberg, “Linkage identification by non-monotonicity detection for overlapping functions,” Evolutionary Computation, Vol. 7, No. 4, 1999, pp. 377–398.

[11] M. Munetomo, “Linkage identification with epistasis measure considering monotonicity conditions,” Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, 2002, pp. 550-554.

[12] A. Fischer and C. Igel, “An Introduction to Restricted Boltzmann Machines,” Proc. CIARP 2012, LNCS 7441, 2012, pp. 14-36.

[13] R. Salakhutdinov and G. E. Hinton, “Deep Boltzmann machines,” *Proc. JMLR Workshop and Conference Proceedings: AISTATS 2009*, vol. 5, 2009, pp. 448-455.

 ISBN: 978-1-61208-461-9