Intruder Data Classification Using GM-SOM

Petr Gajdoš, Pavel Moravec

▶ **To cite this version:**

Petr Gajdoš, Pavel Moravec. Intruder Data Classification Using GM-SOM. 11th International Conference on Computer Information Systems and Industrial Management (CISIM), Sep 2012, Venice, Italy. pp.92-100, 10.1007/978-3-642-33260-9_7 . hal-01551714

HAL Id: hal-01551714
https://inria.hal.science/hal-01551714

Submitted on 30 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Intruder Data Classification using GM-SOM

Petr Gajdoš and Pavel Moravec

Department of Computer Science, FEECS, VŠB – Technical University of Ostrava,
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{petr.gajdos, pavel.moravec}@vsb.cz

Abstract. This paper uses a simple modification of classic Kohonen network (SOM), which allows parallel processing of input data vectors or partitioning the problem in case of insufficient resources (memory, disc space, etc.) to process all input vectors at once. The algorithm has been implemented to meet a specification of modern multicore graphics processors to achieve massive parallelism. The algorithm pre-selects potential centroids of data clusters and uses them as weight vectors in the final SOM network.In this paper, the algorithm is used on a well-known KDD Cup 1999 intruders dataset.

Keywords: SOM; Kohonen Network; parallel computation; KDD Cup 1999 Data Set

1 Introduction

With the massive boom of GPU-based calculations, massive parallelism, memory considerations, simplicity of algorithms and CPU-GPU interaction have yet again to play an important role. In this paper, we present a simple modification of classic Kohonen's self-organizing maps (*SOM*), which allows us to dynamically scale the computation to fully utilize the GPU-based approach.

There were some attempts to introduce parallelism in Kohonen networks [1,2,3,4,5], however we needed an approach which is simple and easy to implement. Moreover, it should work both with and without the bulk-loading algorithm [6].

In this paper, we present such approach, which divides the training set into several subsets and calculates the weights in multi-step approach. Calculated weights with nonzero number of hits serve as input vectors of SOM network in the following step. Presently, we use a two-step approach, however more steps could be used if necessary.

The paper is organized as follows: in second chapter we mention classic SOM networks and describe the basic variant we have used. In third chapter we describe our approach and provide the calculation algorithm, in fourth the GPU-based processing. The fifth chapter introduces experimental data we have used and the final chapter before conclusion presents the comparison of results provided by our method with classic SOM calculation.

2 Kohonen self-organizing neural network

In following paragraphs, we will shortly describe the Kohonen self-organizing neural networks (self-organizing maps – *SOM*). The first self-organizing networks were proposed in the beginning of 70's by Malsburg and his successor Willshaw. SOM was

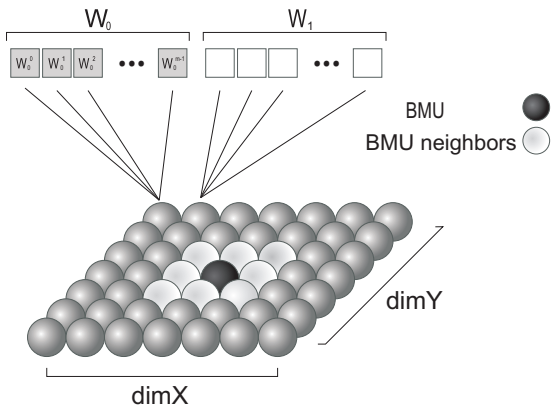

Fig. 1. Kohonen network structure

proposed by Teuvo Kohonen in in the early 1980s and has been improved by his team since. The summary of this method can be found in [7].

The self-organizing map is one of the common approaches on how to represent and visualize data and how to map the original dimensionality and structure of the input space onto another – usually lower-dimensional – structure in the output space.

The basic idea of SOM is based on the human brain, which uses internal 2D or 3D representation of information. We can imagine the input data to be transformed to vectors, which are recorded in neural network. Most neurons in cortex are organized in 2D. Only the adjacent neurons are interconnected.

Besides of the input layer is in SOM only the output (competitive) layer. The number of inputs is equal to the dimension of input space. Every input is connected with each neuron in the grid, which is also an output (each neuron in grid is a component in output vector). With growing number of output neurons, the quality coverage of input space grows, but so does computation time.

SOM can be used as a classification or clustering tool that can find clusters of input data which are more closer to each other.

All experiments and examples in this paper respect following specification of the SOM (see also the Figure 1):

- The SOM is initialized as a network of fixed topology. The variables $dimX$ and $dimY$ are dimensions of such 2-dimensional topology.
- V^m represents an m-dimensional input vector.
- W^m represents an m-dimensional weight vector.
- The number of neurons is defined as $N = dimX * dimY$ and every neuron $n \in < 0, N-1 >$ has its weight vector W_n^m
- The neighborhood radius r is initialized to the value $min(dimX, dimY)/2$ and will be systematically reduced to a unit distance.
- All weights vectors are updated after particular input vector is processed.
- The number of epochs e is know at the beginning.

The Kohonen algorithm is defined as follows:

1. **Network initialization**
   All weights are preset to a random or pre-calculated value. The learning factor η, $0 < \eta < 1$, which determines the speed of weight adaptation is set to a value slightly less than 1 and monotonically decreases to zero during learning process. So the weight adaptation is fastest in the beginning, being quite slow in the end.
2. **Learning of input vector**
   Introduce k training input vectors $V_1, V_2, \ldots, V_k$, which are introduced in random order.
3. **Distance calculation**
   An neighborhood is defined around each neuron whose weights are going to change, if the neuron is selected in competition. Size, shape and the degree of influence of the neighborhood are parameters of the network and the last two decrease during the learning algorithm.
4. **Choice of closest neuron**
   We select the closest neuron for introduced input.
5. **Weight adjustment**
   The weights of closest neuron and its neighborhood will be adapted as follows:

   $$W_{ij}(t+1) = W_{ij}(t) + \eta(t)h(v,t)(V_i - W_{ij}(t)),$$

   where $i = 1, 2, \ldots, dimX$ a $j = 1, 2, \ldots, dimY$ and the radius r of neuron's local neighborhood is determined by adaptation function $h(v)$.
6. **Go back to point 2 until the number of epochs e is reached.**

To obtain the best organization of neurons to clusters, a big neighborhood and a big influence of introduced input are chosen in the beginning. Then the primary clusters arise and the neighborhood and learning factor are reduced. Also the $\eta \to 0$, so the changes become less significant with each iteration.

3 GM-SOM method

The main steps of SOM computation have already been described above. Following text is focused on description of proposed method, that in the end leads to results similar to the classic SOM (See also Figure 2 for illustration of our approach). We named the method Global-Merged SOM, which suggests, that the computation is divided into parts and then merged to obtain the expected result. Following steps describe the whole process of GM-SOM:

1. **Splitting of input set** The set of input vectors is divided into a given number of parts. The precision of proposed method increases with the number of parts, however, it has own disadvantages related to larger set of vectors in the final phase of computation process. Thus the number of parts will be usually determined from the number of input vectors. Generally, $k \gg N * p$, where k is the number of input vector, N is the number of neurons and p is the number of parts. The mapping of input vectors into individual parts does not affect final result. This will be later demonstrated by the experiments, where all the input vectors were either split sequentially (images) or randomly.

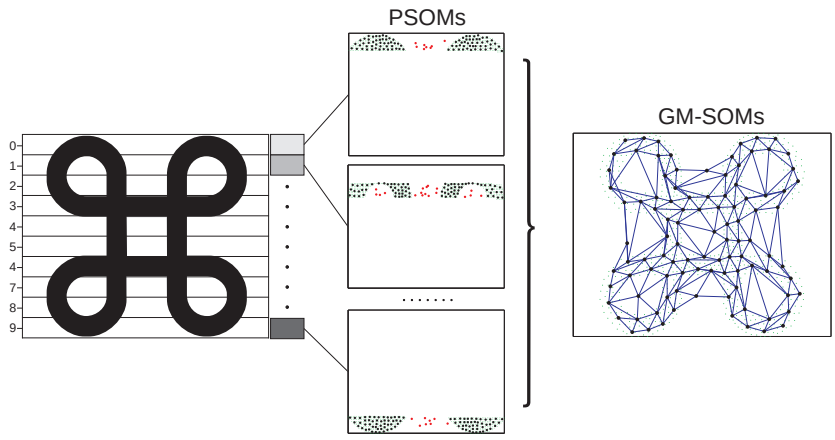

Fig. 2. GM-SOM: An Illustrative schema of the proposed method. All input vectors are divided into ten parts in this case.

2. **In parts computation** Classic SOM method is applied on every part. For simplicity sake, an acronym *PSOM* will be used from now on to indicate SOM, which is computed on a given part. All PSOMs start with the same setting (the first distribution of weights vectors, number of neurons, etc.) Such division speeds up parallel computation of PSOMs on GPU. Moreover, the number of epochs can be lower than the the number of epochs used for processing of input set by one SOM. This is represented by a factor f, which is going to be set to $\frac{1}{3}$ in our experiments.
3. **Merging of parts** Weight vectors, that where computed for each part and correspond to neurons with at least one hit, represent input vectors in the final phase of GM-SOM. The unused neurons and their weight vectors have light gray color in Figure 2. A merged SOM with the same setting is computed and output weights vectors make the final result of proposed method.

The main difference between the proposed algorithm and well known batch SOM algorithms is, that individual parts are fully independent on each other and they update different PSOMs. Moreover, different SOM algorithms can be applied on PSOM of a given part, which makes the proposed algorithm more variable. Next advantage can be seen in different settings of PSOMs. Thus more dense neuron network can be used in case of larger input set. The last advantage consists in a possibility of incremental updating of GM-SOM. Any additional set of input vectors will be processed by a new PSOM in a separate part and the final SOM will be re-learnt. For interpretation see Figure 3.

4 GPU Computing

Modern graphics hardware plays an important role in the area of parallel computing. Graphics cards have been used to accelerate gaming and 3D graphics applications, but

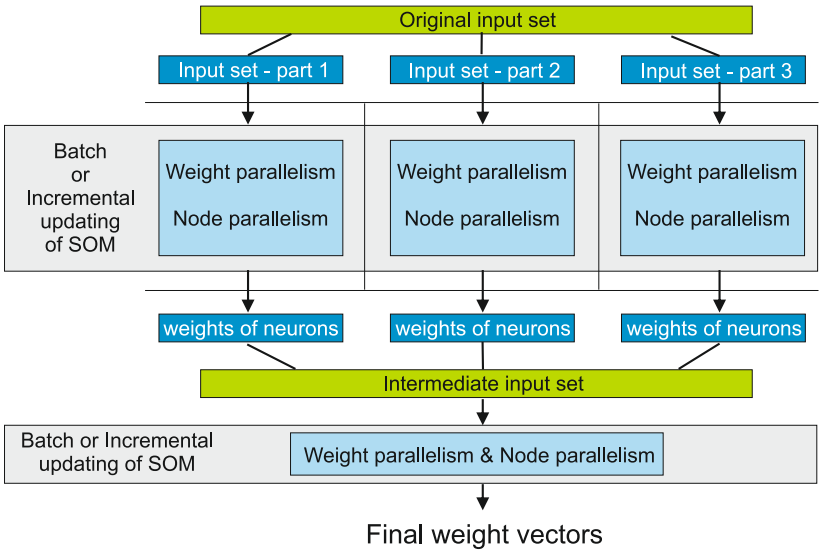

Fig. 3. GM-SOM: Parallelization of the SOM computation by proposed method

recently, they have been used to accelerate computations for relatively remote topics, e.g. remote sensing, environmental monitoring, business forecasting, medical applications or physical simulations etc. Architecture of GPUs (Graphics Processing Unit) is suitable for vector and matrix algebra operations, which leads to a wide use of GPUs in the area of information retrieval, data mining, image processing, data compression, etc. Nowadays, the programmer does not need to be an expert in graphics hardware because of existence of various APIs (Application Programming Interface), which help programmers to implement their software faster. Nevertheless, it will be always necessary to follow basic rules of GPU programming to write a more efficient code.

Four main APIs exists today. The first two are vendor specific, i.e. they were developed by two main GPU producers - AMD/ATI and nVidia. The API developed by AMD/ATI is called ATI Stream and the API developed by nVidia is called nVidia CUDA (Compute Unified Device Architecture). Both APIs are able to provide similar results. The remaining two APIs are universal. The first one was designed by Khronos Group and it is called OpenCL (Open Computing Language) and the second was designed by Microsoft as a part of DirectX and it is called Direct Compute. All APIs provide a general purpose parallel computing architectures that leverages the parallel computation engine in graphics processing units.

The main advantage of GPU is its structure. Standard CPUs (central processing units) contain usually 1-4 complex computational cores, registers and large cache memory. GPUs contain up to several hundreds of simplified execution cores grouped into so-called multiprocessors. Every SIMD (Single Instruction Multiple Data) multiprocessor drives eight arithmetic logic units (ALU) which process the data, thus each ALU of a

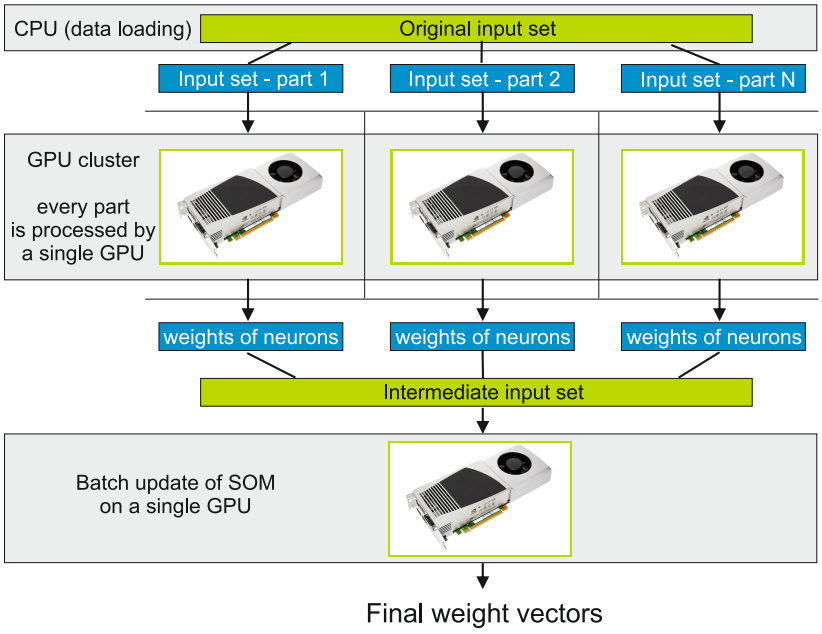

Fig. 4. GM-SOM: Parallelization of the SOM computation by proposed method

multiprocessor executes the same operations on different data, lying in the registers. In contrast to standard CPUs which can reschedule operations (out-of-order execution), the selected GPU is an in-order architecture. This drawback is overcome by using multiple threads as described by Wellein et al.[8]. Current general-purpose CPUs with clock rates of 3 GHz outperform a single ALU of the multiprocessors with its rather slow 1.3 GHz. The huge number of parallel processors on a single chip compensates this drawback.

The GPU computing has been used in many areas. Andrecut [9] described CUDA-based computing for two variants of Principal Component Analysis (PCA). The usage of parallel computing improved efficiency of the algorithm more than 12 times in comparison with CPU. Preis et al. [10] applied GPU on methods of fluctuation analysis, which includes determination of scaling behavior of a particular stochastic process and equilibrium autocorrelation function in financial markets. The computation was more than 80 times faster than the previous version running on CPU. Patnaik et al. [11] used GPU in the area of temporal data mining in neuroscience. They analyzed spike train data with the aid of a novel frequent episode discovery algorithm, achieving a more than 430$\times$ speedup. The GPU computation has also already been used in intrusion detection systems [12] and for human iris identification (for our experiments on this topic see [13]).

The use of our SOM modification on GPU is illustrated in Figure 4.

5 Data Collection

The KDD Cup 1999 collection [14] represents data indicating several basic types of attack on computer networks. Significant features have been recorded and the type of attack has been evaluated. The data is based on the 1998 DARPA Intrusion Detection Evaluation Program, which has been prepared and managed by MIT Lincoln Labs. The main task is to build a predictive model (i.e. a classifier) capable of distinguishing between intrusions (attacks), and normal connections.

The 1999 KDD intrusion detection contest dataset used a specific format of features, extracted from original raw data, which corresponded to 7 weeks of traffic and had approximately 4 GiB of compressed data streams, which were processed into about five million connection records.

Each of the records consists of approximately 100 bytes and the result is labeled based on the actual attack type (or normal, in case it is part of normal traffic) which falls into one of the four basic categories:

- *DOS* – denial-of-service – the machine is flooded with requests and the exhaustion of resources is attempted. Nowadays, it is mostly used in the distributed form.
- *R2L* – remote to local – unauthorized access from a remote machine, e.g. trying to guess passwords or SSH keys.
- *U2R* – user to root – unauthorized user rights elevation, typically through some exploit (e.g. using buffer overflow techniques) to gain superuser (root) privileges.
- *probing* – port scans, active network surveillance and other probing.

There are 22 defined attack types, which were typical in 1998 and the normal category in the training data and some additional attack types present only in the testing data, which the classification framework should detect. The records contain both the categorical, Boolean and numerical attributes, which makes the classification harder.

6 Experiments

The Table 1 is a summary table for seven performed experiments. The whole training data collection was divided into several parts for individual experiments. In each experiment, every part represented an input data set for SOM. Then all partial results (weights of neurons) were merged together and became a new input set for final computation of SOM. Thus the total time consists of particular computation times of SOMs plus required merging time. Sufficient number of graphics processor units (GPU) are required for such computation. Usually, more graphics contexts can be created on a single GPU, however, they share all resources (memory, multiprocessors, etc.). If the number of GPUs is less than the number of parts, and just a single graphics context per GPU can be created (because of memory requirements, etc.), more parts must be computed in series. Then the computation times of all such parts must be summed. Therefore, the column "Total time" of the Table 1 shows ideal computation times in case of sufficient computation power.

The number of parts affects the precision of SOM classification. The higher number of parts the smaller precision. It depends on process of division of input data set as

Table 1. Computation times with respect to subdividing of input data set.

exp. id	parts [#]	size of a part [#]	Comp. time per part [s]	Merging time [s]	Total time [s]	Precision [%]
1	1	1469529	1249	0	1249	94,29
2	5	293906	249	4	453	94,13
3	10	146953	124	9	133	93,57
4	20	73477	62	18	80	90,72
5	30	48985	43	27	70	88,47
6	40	36739	32	36	68	87,91
7	50	29391	26	45	71	87,34

well. Every part consists of random set of input vectors in our experiments. Every two parts represent disjoint sets of input vectors. Finally, every vector of the original data set (non-divided) belongs to some part. The ratio between the number of parts and final precision of classification method will be the subject of further research.

7 Conclusion

The need of parallel computation of SOM drove us to a new method, that has been also utilized in this paper. Although it has some common features with well known SOM batch or hierarchical algorithms, it is not one of them, as it has its unique properties. The results presented in previous section show that whilst the classic SOM is much faster because of the GPU-based computation, the use of PSOMs and their computation by separate nodes furher improves the computation time.

Firstly, the proposed algorithm can utilize the power of batch processing in all inner parts (PSOMs). Secondly, all PSOMs can have different number of neurons in their networks, which could be found in hierarchical algorithms. Lastly, our method excludes neurons, which do not cover any input vectors in the intermediate phase of GM-SOM.

All experiments suggest, that the results are very close to results provided by classic SOM algorithm. Also, since we have used the KDD Cup 1999 collection, the data is comparable with other experiments done on this collection.

Acknowledgment

This work was supported by the Bio-Inspired Methods: research, development and knowledge transfer project, reg. no. CZ.1.07/2.3.00/20.0073 funded by Operational Programme Education for Competitiveness, co-financed by ESF and state budget of the Czech Republic.

References

1. Mann, R., Haykin, S.: A parallel implementation of Kohonen's feature maps on the warp systolic computer. In: Proc. IJCNN-90-WASH-DC, Int. Joint Conf. on Neural Networks. Volume II., Hillsdale, NJ, Lawrence Erlbaum (1990) 84–87
2. Openshaw, S., Turton, I.: A parallel Kohonen algorithm for the classification of large spatial datasets. Computers & Geosciences **22**(9) (November 1996) 1019–1026
3. Nordström, T.: Designing parallel computers for self organizing maps. In: Forth Swedish Workshop on Computer System Architecture. (1992)
4. Valova, I., Szer, D., Gueorguieva, N., Buer, A.: A parallel growing architecture for self-organizing maps with unsupervised learning. Neurocomputing **68** (2005) 177 – 195
5. Wei-gang, L.: A study of parallel self-organizing map. In: Proceedings of the International Joint Conference on Neural Networks. (1999)
6. Fort, J., Letremy, P., Cottrel, M.: Advantages and drawbacks of the batch Kohonen algorithm. In: 10th-European-Symposium on Artificial Neural Networks, Esann'2002 Proceedings. (2002) 223–230
7. Kohonen, T.: Self-Organizing Maps. second (extended) edn. Springer Verlag, Berlin (1997)
8. Hager, G., Zeiser, T., Wellein, G.: Data access optimizations for highly threaded multi-core CPUs with multiple memory controllers. In: IPDPS, IEEE (2008) 1–7
9. Andrecut, M.: Parallel GPU implementation of iterative PCA algorithms. Journal of Computational Biology **16**(11) (November 2009) 1593–1599
10. Preis, T., Virnau, P., Paul, W., Schneider, J.J.: Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets. New Journal of Physics **11**(9) (2009) 093024 (21pp)
11. Patnaik, D., Ponce, S.P., Cao, Y., Ramakrishnan, N.: Accelerator-oriented algorithm transformation for temporal data mining. CoRR **abs/0905.2203** (2009)
12. Platos, J., Kromer, P., Snasel, V., Abraham, A.: Scaling IDS construction based on non-negative matrix factorization using GPU computing. In: Information Assurance and Security (IAS), 2010 Sixth International Conference on. (aug. 2010) 86–91
13. Gajdos, P., Platos, J., Moravec, P.: Iris recognition on GPU with the usage of non-negative matrix factorization. In: Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on. (29 2010-dec. 1 2010) 894–899
14. Stolfo, S., Fan, W., Lee, W., Prodromidis, A., Chan, P.: Cost-based modeling for fraud and intrusion detection: results from the jam project. In: DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings. Volume 2. (2000) 130 –144 vol.2