Available online at www.sciencedirect.com
ScienceDirect
Procedia Engineering 69 (2014) 1499 – 1508
Procedia Engineering
www.elsevier.com/locate/procedia

24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013

A Machine Learning Approach for Abstraction Based on the Idea of Deep Belief Artificial Neural Networks

Florian Neukart*, Sorin-Aurel Moraru

University of Brasov, B-dul Eroilor nr.29, 500036 Brasov, Romania

Abstract

In a time-critical world knowledge at the right time might decide everything. However, storing data does not correspond with understanding the knowledge it contains. Thus, solutions capable of learning problem statements and gathering knowledge from huge amounts of data, be it structured or unstructured, are required. This is where computational intelligence and the introduced approach apply: within this paper, a new method of combining restricted Boltzmann machines and feed forward artificial neural networks is elucidated as well as the accuracy of the resulting solution is proofed.

Keywords: Computational Neuroscience; Computational Intelligence; Data Mining; Artificial Neural Networks

1. Introduction

A significant need exists for techniques and tools with the ability to intelligently assist humans in analyzing very large collections of data in search of useful knowledge; in this sense, the area of data mining (DM) has received special attention [1]. Data mining is the process of discovering valuable information from observational data sets, which is an interdisciplinary field bringing together techniques from databases, machine learning, optimization theory, statistics, pattern recognition, and visualization [2]. As the term indicates, practically applied data mining functions and algorithms are the tools to detect (mine) coherencies between data out of unmanageable amounts of raw data. DM finds application in several fields of industry, science or even daily life, as some examples are customer acquisition, prediction of weather or the evaluation of user-specific click paths in online portals for offering the user products of interest. Thus, the aim of data mining is to extract knowledge out of huge amounts of

* Corresponding author. Tel.: +49 171 12 199 12.
E-mail address: florian.neukart@edu.campus02.at

doi:10.1016/j.proeng.2014.03.147

data. Besides conventional, statistical functions as correlation and regression, methods from signal theory, pattern recognition, clustering, computational neuroscience, fuzzy systems, evolutionary algorithms, swarm intelligence and machine learning are applied [3]. Especially computationally intelligent systems allow the implementation of pattern recognition, clustering and learning by the application of evolutionary algorithms and paradigms of computational neuroscience. Computational intelligence (CI) is a subfield of artificial intelligence (AI) and needs further explanation.AI as a whole tries to make systems behave like humans do, whereas CI relies on evolutionary approaches to solve, amongst others, problems suitable for computers, like detecting similarities in huge amounts of data or optimization problems. Within the field of AI robotics, CI approaches find application for ensuring robust control, planning, and decision making [4,5,6,7,8]. CI techniques have experienced tremendous theoretical growth over the past few decades and have also earned popularity in application areas e.g. control, search and optimization, data mining, knowledge representation, signal processing, and robotics [9].However, when talking about the application of CI-paradigms in the further chapters, it has to be clearly defined which meaning the umbrella term CI bears in the field of DM at first. The term itself is highly controversial in research and used in different manners. Generally, Fulcher et al. and Karplus provide widely accepted definitions, which are also suitable within the context of this elaboration:

- Nature-inspired method(s) + real-world (training) data = computational intelligence [10].
- CI substitutes intensive computation for insight into how the system works. Neural networks, fuzzy systems and evolutionary computation were all shunned by classical system and control theorists. CI umbrellas and unifies these and other revolutionary methods [11].
- Another definition of computational intelligence emphasizes the ability to learn, to deal with new situations, and to reason [12].

2. The SHOCID project, fundamentals, and actual research

The introduced approach successfully finds application in a prototypical DM software system named SHOCID [13,14,15,16,17]. SHOCID is a generic, knowledge-based neurocomputing [18] (KBN) software system, and the acronym for „System applying High Order Computational Intelligence" in data mining (SHOCID), applying computational intelligence-paradigms, methods and techniques in the field of data mining. KBN and in the sequel SHOCID concerns the use and representation of symbolic knowledge within the neurocomputing paradigm [19, 20]. For being able to understand the introduced approach, one is required to be equipped with some basic knowledge of two special kinds of artificial neural networks (ANNs) Geoffrey Hinton [22,23] has made considerable research efforts on:

- Boltzmann machines and
- deep belief networks.

The first ones belong to the class of fully connected ANNs: each of the neurons of such an ANN features connections to every other neuron, except themselves – no self-connections exist, but as many input as output ones. Thus, a fully connected ANN has no processing direction like a feed forward ANN has. Two very popular and simple fully connected ANNs are the

- Hopfield network [24] and the
- Boltzmann machine.

Especially the latter type is of utmost importance for SHOCID, as it is the foundation the system's deep belief networks for classification. Both network types are presented an initial pattern through its input neurons, which does not differ from MLPs. However, according to the weight calculations, a new pattern is received from the output neurons. The difference to other ANNs now is that this pattern is fed back to the input neurons, which is possible, as the input and output layer are the same. This cycle continues as long as it takes the network to stabilize. Until stabilization, the network is in a new state after an iteration, which is used for comparison of the actual pattern and the input vector [25]. Both Hopfield ANNs and Boltzmann machines belong to the class of thermal ANNs, as they feature a stochastic component, the energy function, which is applied similarly to simulated annealing-learning. Energy-based probabilistic models define a probability distribution through an energy function, as follows:

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{1}$$

where Z is the normalization factor and called the partition function:

$$Z = \sum_{x} e^{-E(x)} \tag{2}$$

An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data.

3. SHOCID deep belief-like ANN

SHOCID makes use of restricted Boltzmann machines [29] for preparing ANN structures for classification tasks within the field of Data Mining. Stacked, restricted Boltzmann machines (RBM) are called deep belief networks [23], which learn to extract a deep hierarchical representation of the training data. However, the Boltzmann machines used for SHOCID are standard ones and follow the same purpose RBMs in DBNs do. The introduced DB-like approach does not directly stack together Boltzmann machines, but extracts the weights gathered from training for inducing these into a feed-forward ANN. Summing up, this works by normalizing the presented data both bipolar and multiplicative, the output of the former one used for training the Boltzmann machines, the latter one for training the feed forward ANN. The structure of the DB-like ANNs does not differ from a common feed-forward ANN, as the only difference is that the weight initialization does not happen at random, but by extracting the weights from pre-trained Boltzmann machines. The following figure helps to understand the proceeding (Fig. 1. DB-like ANN):

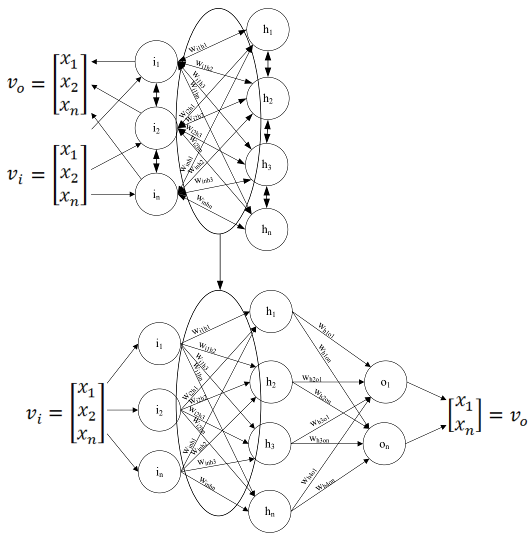

Fig. 1. DB-like ANN.

The weight initialization for the multi-layer perceptron does not happen randomly or by any other known initialization techniques like e.g. Nguyen-Widrow-randomization. A DBN with l layers models the joint distribution between an observed vector x and its l hidden layers hk as follows:

$$P(x, h^1, \ldots, h^l) = \left(\prod_{k=0}^{l-2} P(h^k | h^{k+1})\right) P(h^{l-1}, h^l) \tag{3}$$

wherex $= \mathrm{h}^0, \mathrm{P}(\mathrm{h}^{\mathrm{k}-1}|\mathrm{h}^{\mathrm{k}})$ is a visible-given-hidden conditional distribution in an RBM associated with level k of the DBN, and $P(h^{k-1}|h^k)$ is the joint distribution in the top-level RBM. The conditional distributions $P(h^k|h^{k+1})$ and the top-level joint (an RBM) $P(h^{l-1}|h^l)$ define the generative model [32]. This does not differ from the introduced approach; the difference is to be found in the processing, namely the feature mapping and weight shifting.

4. Processing

Usually, when dealing with deep belief networks, many hidden layers can be learned efficiently by composing restricted Boltzmann machines, using the feature activations of one as the training data for the next [30]. The DB-like ANNs differ from that as they copy the weights into of the pre-trained Boltzmann machine to an MLP. Moreover, DB-like ANNs (= the MLPs with the Boltzmann machine's weights) apply simulated annealing within SHOCID, as the Boltzmann machine itself is a thermal network as well. Thus, SA is the first choice as a learning strategy and has furthermore proved to perform well. The processing within the SHOCID DB-like ANNs can be split up in six major steps:

- Data normalization for the Boltzmann machine
- Boltzmann processing
- Weight normalization
- Weight shifting
- Data normalization for the MLP
- MLP processing

4.1. Data normalization for the Boltzmann machine

For the Boltzmann machine the normalization must output bipolar values. Thus, as a first step the input vector is normalized range-mapped, between -1 and 1, which produces a floating point value for each input value not larger than 1 and not smaller than -1:

$$f(x) = \left(\left(\frac{x - min}{x - max}\right) * (high - low)\right) + low \tag{4}$$

Breakdown:
x = The value to normalize
min = The minimum value x will ever reach
max = The maximum value that x will ever reach
low = The low value of the range to normalize into (typically -1 or 0)
high = The high value of the range to normalize into (typically 0 or 1)

As the equation shows, mapping allows to map one numeric range to another. For this, a data mining system has to perform not only the step of normalization, but also parse through the data to collect the maximum and minimum values x (the value to normalize) will ever reach. However, not the floating point values are then fed to the Boltzmann machine, but the following output:

$$x_n = \begin{cases} -1 \quad , & if\left(\left(\frac{x - min}{x - max}\right) * (high - low)\right) + low < 0 \\ 1 \quad , & \left(\left(\frac{x - min}{x - max}\right) * (high - low)\right) + low > 0 \end{cases} \tag{5}$$

4.2. Boltzmann processing

In the first iteration, SHOCID creates a Boltzmann machine with the input layer size equal to the size of the number of the input vector. The input vector may or may not be a multidimensional array. Independent from the training data, which may also contain the values of the output neuron(s), SHOCID is able to determine the difference as it requires the user to enter the number of input neurons. The number of hidden neurons represents the number of features within the Boltzmann machine. Thus, within the first Boltzmann machine the number of features equals the size of the input vector. The first Boltzmann machine is then trained with every input vector of the training data with the target to auto associate the input vector's states. The higher the number of input vectors, the more unlikely it is that even one of the input vectors can be associated correctly. However, after the training has finished, the weighs of the current Boltzmann machine are copied into a multidimensional array at position 0 – meaning that these will be the weights from the input layer to the first hidden layer in the MLP. Furthermore, all associated output vectors have been stored into an array at this time, as these will serve as inputs for the next Boltzmann Machine. This process is repeated as often as the number of the network's hidden layers has been reached. Therefore, if an MLP requires 3 hidden layers, SHOCID creates 3 Boltzmann machines and saves the weights 3 times. Commonly, the input from one RBM to the next level-one may either be the mean activations $p(h^{(1)} = 1|h^{(0)})$ or samples of $p(h^{(1)}|h^{(0)})$, where $h^{(0)}$ is the visible input RBM and $h^{(0)}$ the RBM above. SHOCID applies the latter one, as the input for an RBM is the associated output the RBM below [28]. The first RBM forwards the associated output it generates from the original input vector. However, the last Boltzmann machine, which is used to determine the weights from the last hidden layer to the output layer of the MLP, uses the number of output neurons as the number of features to detect. The output vector may or may not be a multidimensional array as well and is of no importance until the last Boltzmann machine has been created.

4.3. Weight normalization

In difference to all introduced approaches, not only the input values of the artificial neural network are normalized, but also the weights of a trained Boltzmann machine. This is, as especially when a (restricted) Boltzmann machine is required to deal with numerous and manifold datasets made up of multiple attributes, its final weights tend to differ from each other to a large extent. Therefore, multiplicative normalization (equation 5) is applied on the Boltzmann machine's weights to scale them to proper values. Verification has shown that multiplicative normalization of the weights results in a dramatic increase of learning performance during training.

$$f = \frac{1}{\sqrt[2]{\sum_{i=0}^{n-1} x_i^2}} \tag{6}$$

calculates the weight normalization factor f where x is the weight on position i of the weight list.

4.4. Weight shifting

Weight shifting is the process of extracting the normalized weights from the restricted Boltzmann machine and inserting them into a newly created layer within a feed forward artificial neural network.

4.5. Data normalization for the MLP

Normalization within the MLP happens multiplicative, as per default within all MLPs of SHOCID:

$$f = \frac{1}{\sqrt[2]{\sum_{i=0}^{n-1} x_i^2}} \tag{7}$$

where$x$ is the value to normalize.

4.6. MLP processing

As a last step the MLP is created according to the required number of layers and neurons. Each hidden layer has the same number of neurons as the input layer, as within the Boltzmann pre-training each value of an input vector was used to create one feature.

5. Purpose

Empiric research has shown that although the auto association of the Boltzmann machines does not reproduce exactly the presented input values, the copying of the weights of these pre-trained networks to the final MLP does intensely decrease the number of training iterations needed. Furthermore, such networks do not require a high number of neurons per hidden layer for being able to finish training successful, although the network architecture may be of considerable depth. Moreover, the research has shown that such networks are very successful in processing datasets they have not been trained for.

6. Algorithm

Furthermore, as simulated annealing has proofed to come to the most accurate solutions in short time within cortical ANNs, SHOCID's algorithm for evolving cortical ANNs is as follows:

Start

1. Repeat

a) Creation of a Boltzmann machine B_n featuring n_{input} neurons in the input layer and n_{input} neurons in the hidden layer, and an arraylist $l_{weig\,hts}$ for caching the weights before the weight shift.

i. Initial selection of RBM weights at random.

ii. ***Repeat***

1. Set the input neurons' states to the learning vector x, while $x = x(t) = [x0(t), x1(t), \ldots, xn(t)]T, t = 1, 2, \ldots$

2. For each
hidden neuron j connected to the input neuron i
set states according to

$$x_t = h_1\left(w_{0i} + \sum_j w_{ij} x_j\right)$$

3. For each
edge e_{ij} calculate $\mathrm{Positive}(e_{ij}) = x_i * x_j$

4. For each
input neuron i connected to the hidden neuron j set states according to

$$x_t = h_1\left(w_{0j} + \sum_i w_{ji} x_i\right)$$

5. For each
hidden neuron j connected to the input neuron i set states according to

$$x_t = h_1\left(w_{0i} + \sum_j w_{ij} x_j\right)$$

6. For each
edge e_{ij}

a. Calculate $\mathrm{Negative}(e_{ij}) = x_i * x_j$
b. Update weight of e_{ij} according to
c. $w_{ij} = w_{ij} + \mu(\mathrm{Positive}(e_{ij}) - \mathrm{Negative}(e_{ij}))$
d. Add w_{ij} to $l_{weights}$

iii. ***Until criteria are reached***

b) Calculate the weight normalization factor f with

$$f = \frac{1}{\sqrt[2]{\sum_{i=0}^{n-1} w_i^2}}$$

c) ***For each***
weight in $l_{weights}$

i. $w_{norm} = l_{weights\ i} * f$
ii. Replace $l_{weights\ i}$ with w_{norm}
iii. Copy the weights of B_n in the weights array.

d) Shift the weights

2. Until the number of hidden layers nh has been reached.

3. Creation of the final Boltzmann machine B_f featuring n_{input} neurons in the input layer and noutput neurons in the hidden layer.
4. Auto associate each input vector of the training data.
5. Copy the weights of B_n in the weights array.
6. Creation of the MLP M with n_{input} neurons as input neurons, n_h hidden layers each featuring n_{input} neurons and n_{output} neurons in the output layer.

7. Repeat

a) Calculate the output of M for the value $d \in D$
b) Evaluate the fitness of each neuron:
c) Calculate the error $\delta_{d \in D}$ for each output neuron $d \in D$

$$\delta_{d \in D =} (desired_{d \in D} - actual_{d \in D}) actual_{d \in D} (1 - actual_{d \in D})$$

d) Calculate the error δ_{hid} for each hidden neuron hid

$$\delta_{hid} = actual_{hid}(1 - actual_{hid}) \sum_{d \in D} (w_{hidd \in D} \delta_{d \in D})$$

e) ***Repeat***

i. Create new ANN and randomize weights according to T

ii. Calculate the error $\delta_{d \in D}$ for each output neuron $d \in D$

$$\delta_{d \in D} = (desired_{d \in D} - actual_{d \in D}) actual_{d \in D} (1 - actual_{d \in D})$$

iii. Calculate the error δ_{hid} for each hidden neuron hid

$$\delta_{hid} = actual_{hid}(1 - actual_{hid}) \sum_{d \in D} (w_{hidd \in D} \delta_{d \in D})$$

iv. Compare solutions according to

$$\Delta_{C(S')C(S)} = C(S') - C(S)$$

v. If $C(S')$ is better than $C(S)$, replace $C(S)$

f) ***Until max tries for current temperature reached***

g) Decrease temperature by

$$\varphi_{C(S')C(S)} = T * e^{\frac{ln\left(\frac{S}{e}\right)}{c-1}}$$

8. Until lower temperature bound is reached

End

Algorithm 1 - DB-like ANN

Breakdown:

n_{input}: Number of input neurons
n_{output}: Number of output neurons
n_h: Number of hidden layers
B_i: The current Boltzmann machine
M: The MLP to be trained
$d \in D$: Actual input
$\delta_{d \in D}$: Error for input $d \in D$ at output neuron o_n in the current generation
δ_{hid}: Error for hidden neuron
r_{max}: Allowed RMSE

7. Verification

A financial problem consisting of data that follows consists of black scholes option prices for volatility levels running from 20 percent to 200 percent, for time remaining running from 5 to 15 days, and for strike price running from 70 dollars to 130 dollars. The stock price was set to 100 dollars and the interest rate to 0 percent when generating the data [33]. The number of datasets to learn was 1,530. However, for this second presented problem the elaboration at hand does not contain achieved and ideal outputs, as 1,530 lines for each iteration would then be needed. For gathering detailed information, the SHOCID prototype can be applied. Moreover, not only the RMSE was important to learn, but the created ANN agents have also only been allowed an absolute deviation from the actual learned value to the target value from 5 dollars.

Table 1. General training parameters.

Number of training data sets	Number of verification datasets	Number of input fields	Number of output fields
1,530	100	3	1

The system therefore has to learn the problem provided from scientific consultant services [33]. Furthermore, SHOCID had to normalize each of the inputs as the values have initially not been scaled between 0 and 1. Moreover, the target was not only to train the solution with the provided 1,530 training data sets and have a look at the performance, but also verification on 100 new and not trained datasets, which have been derived from the training datasets by correlation to prove the functioning of the solution. As the verification test results have shown that all of SHOCID's provided solution types are able to solve non-linear problems, the black scholes test results have only been included for the new learning approaches and ANN types as well as for their direct competitors. In case of transgeneticNeuroEvolution the direct competitor is standard simulated annealing learning, as the former one is also based on SA-learning.

7.1. Simulated annealing training 1

The first simulated annealing training has been applied to the black scholes problem to serve as comparator for the DBN-like ANN learning approach, which also makes use of SA learning.

Table 2. Parameters SA 1.

Normalization:	Input fields	Output fields	Hidden Layers	Hidden Neurons	Allowed RMSE	Allowed absolute deviation of target
Yes	3	1	1	5	1 per cent	5

Table 3. Simulated annealing 1.

Run	Training iterations	Final RMSE
1	855	2.3440388409399297E-4
2	755	2.4878078905289343E-4
3	560	2.4221577041141782E-4
4	643	3.2309789223214857E-4
5	764	3.3617530058731646E-4

Solution number 1 has been applied on the test datasets and achieved a classification success of 83 % within the allowed parameters. 17 % of the data have been misclassified with an inaccuracy between 5.22 and 13.48 %. The following picture shows a typical SA-run (run 3), with continuous error improvement and no outliers, compared to transgenetic runs where outliers are very common.

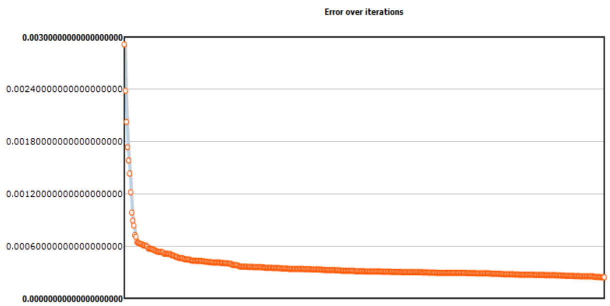

Fig. 2. Simulated Annealing run.

7.2. Simulated annealing training 2

The second simulated annealing training has been applied to the black scholes problem to serve as comparator for DBN-like ANN learning approach as well, which also makes use of SA learning. However, the difference is that the suggested test settings from scientific consultant services [33] have been used in run 1 – 3, which were 22 neurons in the first hidden layer and 8 in the second. Run 4 and 5 received 40 hidden neurons in the second hidden layer, so that more than one configuration of SA solutions serve for comparison.

Table 4. Parameters SA 2.

Normalization	Input fields	Output fields	Hidden Layers	Hidden Neurons Layer 1	Hidden Neurons Layer 2 Run 1 - 3	Hidden Neurons Layer 2 Run 1 - 3	Allowed RMSE	Allowed absolute deviation of target
Yes	3	1	2	22	8	40	1 per cent	5

Table 5. Simulated annealing 2.

Run	Training iterations	Final RMSE
1	363	2.3151416966918833E-4
2	258	2.7189890778372424E-4
3	454	2.41296129499371E-4
4	2,396	2.599023163521219E-4
5	634	3.8e73543294647323E-4

Solution number 1 has been applied on the test datasets and achieved a classification success of 82 % within the allowed parameters. 18 % of the data have been misclassified with an inaccuracy between 0.15 and 12.39 %.

7.3. Deep belief-like ANNs

Deep belief-like ANNs make use of simulated annealing learning, in both its Boltzmann machines and the final MLP.

Table 6. Parameters DBN.

Normalization	Input fields	Output fields	Hidden Layers	Allowed RMSE	Allowed absolute overall error	Allowed absolute deviation of target
Yes	3	1	3	1 per cent	5 per cent	5

Table 7. DB-like ANN.

Run	Training iterations	Final RMSE
1	16	9.966282764783177E-4
2	5	7.864950633377531E-4
3	6	9.565366982744945E-4
4	18	9.397388187292288E-4
5	6	8.5149224962310 8E-4

Solution number 1 has been applied on the test datasets and achieved a classification success of 100 % within the allowed parameters. 0 % of the data have been misclassified.The following chart shows a typical DB-like ANN learning run (run 1), improving its quality quickly without outliers:

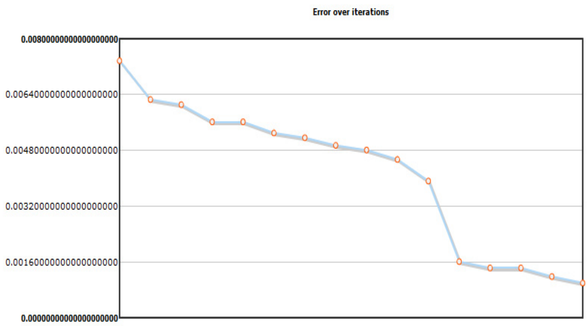

Fig. 3. DB-like ANN run.

8. Conclusion

The verification has shown that deep architectures are suitable for solving complex problem statements in every day-business, as it outperformed MLPs without pre-classification by RBMs, thus purely relying on learning strategies and intelligent weight initialization. Therefore, also complex ANN structures do not longer belong exceptionally to the scientific domain and will find application in data mining solutions in the near future. The tests showed that independent from SHOCID's learning approaches, be it learning through genetic approaches or propagation applied with common feed-forward ANNs, the DB-like ANN not only learned the presented test problem more accurately and in shorter time. Most importantly, the DBN-like approach outperformed any other approach in verification as the classification success on new datasets showed 100 % accuracy in the tests.

References

[1] Nedjah Nadia et al. (2009): *Innovative Applications in Data Mining*; Berlin Heidelberg: Springer-Verlag, p. 47.
[2] Yin Yong, Kaku Ikou, Tang Jiafu, Zhu JianMing (2011): *Data Mining: Concepts, Methods and Applications in Management and Engineering Design (Decision Engineering)*; UK: Springer-Verlag, p. V.
[3] Runkler Thomas A. (2010): *Data Mining - Methoden und Algorithmen intelligenter Datenanalyse*; Wiesbaden: Vieweg+Teubner | GWV Fachverlage GmbH, p. V.
[4] Liu Dikai, Wang Lingfeng, Tan Kay Chen (2009): *Design and Control of Intelligent Robotic Systems*; Berlin Heidelberg: Springer-Verlag, p. 2.
[5] Nolfi Stefano, Floreano Dolfi (2000): *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*; Bradford Books.
[6] Jain Lakhmi C. (1998*): Soft Computing for Intelligent Robotic Systems*: Physica-Verlag.
[7] Watanabe Keigo, Hashem M.M.A. (2004): *Evolutionary Computations: New Algorithms and Their Applications to Evolutionary Robotics*; Heidelberg: Springer-Verlag .
[8] Teshnehlab M., Watanabe K. (1999): *Intelligent Control Based on Flexible Neural Networks (Intelligent Systems, Control and Automation: Science and Engineering)*; Dordrecht: Kluwer Academic Publishers.
[9] Liu Dikai, Wang Lingfeng, Tan Kay Chen (2009): *Design and Control of Intelligent Robotic Systems*; Berlin Heidelberg: Springer-Verlag, p. 2.
[10] Fulcher John (2008): *Computational Intelligence: A Compendium; Berlin Heidelberg*: Springer-Verlag, p. 38.
[11] Karplus W. (1998) cited in: Kaynak Okyay, Zadeh Lofti A., Türksen Burhan (1998): *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*; Berlin: Springer-Verlag.
[12] Eberhart Russel C., Simpson Patrick K., Dobbins Roy (1996): *Computational Intelligence PC Tools*; Boston MA: Academic Press
[13] Neukart Florian, Moraru Sorin-Aurel, Grigorescu Costin-Marius (2011): *High Order Computational Intelligence in Data Mining - A generic approach to systemic intelligent Data Mining*. Proceedings of Speech Technology and Human-Computer Dialogue (SpeD), 2011 6th Conference on, p. 1-9.
[14] Neukart Florian, Moraru Sorin-Aurel, Szakazs-Simon Peter (2011): *Problem-dependent, genetically evolving Data Mining solutions*. Proceedings of DAAAM 2011.
[15] Neukart Florian, Moraru Sorin-Aurel, Grigorescu Costin-Marius, Szakazs-Simon Peter (2012): *Cortical Artificial Neural Networks and their Evolution - Consciousness-inspired Data Mining*. Proceedings of OPTIM 2012.
[16] Neukart Florian, Moraru Sorin-Aurel, Grigorescu Costin-Marius, Szakazs-Simon Peter (2012): *Transgenetic NeuroEvolution*. Proceedings of OPTIM 2012.
[17] Neukart Florian, Moraru Sorin-Aurel, Grigorescu Costin-Marius, Szakazs-Simon Peter (2012), *Artificial Immune System-inspired NeuroEvolution*. Proceedings of DAAM, 2012
[18] Cloete Ian, Zurada Jacek M. (2000): *Knowledge-Based Neurocomputing*; Cambridge: MIT Press
[19] Kolman Eyal, Margaliot Michael (2009): *Knowledge-Based Neurocomputing*: A Fuzzy Logic Approach; Berlin Heidelberg: Springer-Verlag, p. 5
[20] Moulay Aziz-Alaoui, Cyril Bertelle (2009): *From System Complexity to Emergent Properties*; Berlin Heidelberg: Springer-Verlag, p. 212
[21] Neukart Florian et al. (2011): *High Order Computational Intelligence in Data Mining - A generic approach to systemic intelligent Data Mining*; Proceedings of Speech Technology and Human-Computer Dialogue (SpeD), 2011 6th Conference on, p. 1-9, p. 2.
[22] Hinton Geoffrey E., Sejnowski Terrence J. (1983): *Optimal Perceptual Inference*. Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, Washington DC, pp. 448-453.
[23] Hinton Geoffrey E., Salakhutdinov Ruslan R. (2006): *Reducing the dimensionality of data with neural networks*, Science, vol. 313, no. 5786, pp. 504–507, 2006.
[24] Hopfield J. Joseph (1982): *Neural networks and physical systems with emergent collective computational properties*; Proceedings Nat. Acad. Sci. (USA) 79, 2554-2558.
[25] Heaton Jeff (2010): *Programming Neural Networks with Encog 2 in Java; Chesterfield: Heaton Research*, Inc., p. 339
[26] Krengel Ulrich (1988): *Einführung in die Wahrscheinlichkeitstheorie und Statistik*; Braunschweig/Wiesbaden 1988: Verlag Friedrich Vieweg & Sohn
[27] Kreiszig Erwin (1999): *Advanced Engineering Mathematics*, 8th ed.; Singapore: John Wiley & Sons, p. 1107.
[28] Deeplearning.net (2012): *Restricted Boltzmann Machines (RBM)* [2012-15-08]; Deeplearning.net; URL: http://deeplearning.net/tutorial/rbm.html
[29] Smolensky Pavel (1986): *Information processing in dynamical systems: Foundations of harmony theory*. In Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing: Volume 1: Foundations, pages 194-281. MIT Press, Cambridge, MA.
[30] Scholarpedia (2011): *Boltzmann Machine* (2012-15-08); Scholarpedia; URL: http://www.scholarpedia.org/article/Boltzmann_machine#Restricted_Boltzmann_machines
[31] Edwin Chen's Blog (2011): I*ntroduction to Restricted Boltzmann Machines* [2012-28-08]; URL: http://blog.echen.me/2011/18/introduction-to -restricted-boltzmann-machines
[32] Bengio Yoshua (2009): *Learning Deep Architectures for AI; Foundations and Trendsin Machine Learning*, vol. 2, Yoshua Bengio
[33] Scientific Consultant Services (2003): *Neural Network Test Data* [04-04-2012]; URL: http://www.scientific-consultants.com/nnbd.html