ON IMPROVEMENTS OF
COMPETITIVE NETWORKS PRINCIPLES

Dissertation Thesis

by

Dušan Fedorčák

Ostrava, February 2011
Thesis supervisor:

prof. Ing. Ivo Vondrák, CSc.
ivo.vondrak@vsb.cz

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
Czech Republic

http://www.cs.vsb.cz
http://fei.vsb.cz
http://www.vsb.cz

Ph.D. Thesis
Copyright © 2011 Dušan Fedorčák
Abstract

Artificial neural networks are very powerful computational systems and are capable of solving many various problems. They provide a nonlinear input to output mapper, and therefore, applications in classification, feature extraction or pattern recognition may be expected. This dissertation thesis focuses on competitive networks which belong to the unsupervised learning systems family. The aim of this thesis is to present possible improvements to this area and provide necessary experiments which support suggested ideas. First, an introduction to artificial neural networks is given and several common models are stated including basic feed-forward network, Self-organized Feature Map, Growing Neural Gas etc.. Next, several innovative approaches and methods are suggested. In particular, the main contributions of the thesis are as follows:

1. A classifier based on the competitive network is presented. An innovative method for learning is suggested where the competition is not supervised in the standard way but the input signals to the network are altered in order to achieve the supervision.

2. An application for the presented supervised competitive network is presented. The RBF network initialization process is addressed and possible improvements brought by the supervised competition are examined.

3. An application of competitive network in combination with the force-based graph plotting algorithm is presented. The high-dimensional data visualization technique is addressed, advantages of such approach are explored, and several datasets are visualized.

4. A new insight into competitive network principles is given. An innovative physically based model for the competition is stated and various attributes of this model are examined. The model is built above the particle system which is driven by mechanical laws of motion and all competitive principles are transformed into forces affecting particles. The model is tested against several well-known datasets to prove functionality and usability.
Acknowledgments

First of all, I would like to express my gratitude to my dissertation thesis supervisor, prof. Ivo Vondrák, for his leadership and professional guidance during my research. I would like to express thanks to my colleagues from the Media Research Lab for their valuable comments and help in numerous problems. Special thanks go to the Department of Computer Science management for providing most of the funding for my research and publication activities and for maintaining a pleasant working environment.

Finally, my greatest thanks go to my parents for their infinite patience and my girlfriend Nikola for her love and support.
# Contents

Abstract

Acknowledgments

List of Figures

List of Tables

1 Introduction

1.1 Thesis goals

1.2 Thesis structure

2 Artificial neural networks phenomenon

2.1 History of artificial neural networks

2.2 Neurophysiological inspiration

2.3 Goals of artificial neural networks

2.3.1 Supervised learning and unsupervised learning

2.3.2 Other goals of artificial neural networks

2.4 Feed-forward networks

2.4.1 Mathematical model of neuron cell

2.4.2 Network of perceptrons

2.4.3 Sigmoid transfer function

2.4.4 Transfer functions and universal approximation

2.5 Adaptation of MLP network

2.5.1 Backpropagation

2.5.2 Batch learning and incremental learning

2.5.3 Improved backpropagation

2.5.4 Other methods

2.6 Competitive networks

2.6.1 Common model of competitive network

2.6.2 Hard Competitive Learning

2.6.3 Soft competitive networks without fixed dimensionality

2.6.4 Soft competitive networks with fixed dimensionality

2.6.5 Other methods

2.6.6 Transformations of input dataset

2.6.7 Effect of dimension

2.7 Other artificial neural networks
3 Self organization of labeled data

3.1 Supervised competitive learning
   3.1.1 Learning Vector Quantization networks
   3.1.2 Supervised Growing Neural Gas

3.2 Joined Growing Neural Gas
   3.2.1 The idea of joined reference vector
   3.2.2 Properties of joined input space
   3.2.3 Building a classifier – Applying restricted queries
   3.2.4 Class probability prediction

3.3 Experiments
   3.3.1 Concentric spirals dataset
   3.3.2 MNIST database
   3.3.3 Intrusion detection dataset

4 Competitive network as supportive mechanism

4.1 Self organized preprocessing of feed-forward network
   4.1.1 RBF networks
   4.1.2 Adaptation of RBF network
   4.1.3 Three phase training
   4.1.4 Initialization of RBF layer with GNG and JGNG methods
   4.1.5 Initialization of output layer of RBF network
   4.1.6 Experiments

4.2 Visualization of high-dimensional data using GNG
   4.2.1 Importance of visual information
   4.2.2 High-dimensional visualization methods
   4.2.3 Growing Neural Gas
   4.2.4 Particle/spring graph model
   4.2.5 Attributes of force-based layout of GNG graph

4.3 Examples of GNG driven high-dimensional visualization
   4.3.1 Iris plants database
   4.3.2 MNIST dataset
   4.3.3 Intrusion detection dataset

5 Physically based self organization

5.1 Physically based approach
   5.1.1 Simulated annealing & Boltzmann Machines

5.2 Particle/spring system
   5.2.1 Applications of particle/spring system

5.3 Model of physically based competitive network
   5.3.1 Winning neuron problem
   5.3.2 Forces definition

5.4 Physically based learning
   5.4.1 Equilibrium state
   5.4.2 Effect of velocity
   5.4.3 Effect of dimension

5.5 Physically based self-organizing feature map
List of Figures

2.1 Neuron cell diagram ........................................... 5
2.2 Perceptron diagram ......................................... 7
2.3 Perceptron separation ....................................... 8
2.4 MLP network .................................................. 9
2.5 Two layer network example .................................. 9
2.6 Two layer network example - Input spaces ................. 10
2.7 Sigmoid transfer function .................................... 11
2.8 Lateral inhibition in auditory neural network ............. 15
2.9 Hard competitive learning .................................... 17
2.10 Neural Gas algorithm, two-dimensional example .......... 18
2.11 Growing Neural Gas algorithm, two-dimensional example 20
2.12 Input dataset, three-dimensional example ............... 21
2.13 Growing Neural Gas, three-dimensional example ......... 21
2.14 Neighborhood schemes used by the SOM algorithm ...... 22
2.15 Self-organized Feature Map – Square grid schema, two-dimensional example 23
2.16 Relative distance comparison ............................... 25
2.17 Relative distance comparison 2 ............................ 26
3.1 GNG and JGNG model difference ......................... 32
3.2 Class probability – low-dimensional example .......... 33
3.3 Difference between standard LVQ classifier and JGNG classifier ........................................ 35
3.4 Uncertain cases in JGNG driven MNIST dataset .......... 36
3.5 Structure of the JGNG network trained to detect and classify intrusions 37
4.1 Firing mechanism of an RBF unit ......................... 39
4.2 Simple RBF network ......................................... 39
4.3 GNG and JGNG difference 2 ............................... 42
4.4 Trained RBF network ....................................... 43
4.5 Weight prioritisation in the output layer of RBF network 44
4.6 JGNG initialization of RBF network ....................... 45
4.7 Matrix of scatter plots – four-dimensional UCI Iris Plants Database 47
4.8 Parallel and spherical coordinates – four-dimensional UCI iris plants Database 48
4.9 Self-organizing map – iris dataset ......................... 48
4.10 Force-based layout – the twisted-H three-dimensional example 51
4.11 GNG driven visualization – UCI Iris Plants Database 52
4.12 GNG driven visualization – MNIST dataset .............. 53
4.13 GNG driven visualization – Intrusion dataset .......... 54
5.1 Progression of attractive force . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
5.2 Force field visualization – individual cases . . . . . . . . . . . . . . . . . . . . . 61
5.3 Force field visualization – two dimensional dataset . . . . . . . . . . . . . . . . 62
5.4 Distance progress with various learning rules . . . . . . . . . . . . . . . . . . . . 64
5.5 Physically based self organization – 2D dataset . . . . . . . . . . . . . . . . . . . 67
5.6 Similarity measure – physically based learning; 2D dataset . . . . . . . . . . . . . 68
5.7 Physically based SOM – 2D dataset. Two different configurations with different
spring factor were uses for the physically based SOM . . . . . . . . . . . . . . . . 68
5.8 Similarity measure – SOMs; 2D dataset . . . . . . . . . . . . . . . . . . . . . . . . 69
5.9 Progress of distortion error . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
5.10 Similarity measure – Iris dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
5.11 Similarity measure – SOMs; Iris dataset . . . . . . . . . . . . . . . . . . . . . . . 70
5.12 Physically based competition – MNIST dataset . . . . . . . . . . . . . . . . . . . 71
5.13 Physically based SOM – MNIST dataset. Two different configurations with dif-
ferent spring factor were uses for the physically based SOM . . . . . . . . . . . . . 72
5.14 Physically based SOM – Breakable connections . . . . . . . . . . . . . . . . . . . 73

8.1 ANN ToolBox – Screenshot of application main window . . . . . . . . . . . . . . 88
List of Tables

3.1 Experimental results – JGNG method; MNIST dataset .................................................. 36
3.2 Experimental results – JGNG method; Intrusion detection dataset ............................... 37

4.1 Experimental results – RBF initialization process; 2D dataset ..................................... 42
4.2 Experimental results – RBF initialization process; MNIST dataset ............................. 44
4.3 Experimental results – RBF initialization process; Intrusion detection dataset .............. 45
Chapter 1

Introduction

Artificial neural networks are phenomena of modern computer science. Many research groups
are interested in these systems and artificial neural networks are strong computational device by
any mean. With a rapid evolution of computer hardware in parallel it is possible to utilize very
complicated neural systems and there are lots of publications and application studies on this
topic. My motivation for studying this topic was and is great fascination of biologically inspired
systems. These fascination partially raised on attitude that computer science does not have any
sizable subject of research and practically it is based on abstract concepts. On the other side the
computer science is the best place to realize your creativity and these facts directed my studies
to artificial neural networks.

1.1 Thesis goals

This dissertation thesis is a summary of my studies and research in competitive networks and
artificial neural networks in common. The goal of my research is to improve various aspects
of competitive networks. First, the classification problem and according supervised competitive
learning process is addressed. The goal is to utilize information given by supervision in a different
and more natural way than it is common. Second, this new model of supervised competition
is applied to typical tasks where competitive networks are used and possible improvements are
discussed. During my work on these topics, I realized that there is a similarity between the
standard competitive learning and the simulation of physically based particle system. Hence,
another goal of my thesis is to propose a different insight into the paradigm of competitive
networks through the particle system simulation and to examine dual attributes of both systems.

1.2 Thesis structure

The first chapter of my thesis is focused on the history and aim of artificial neural networks
with detailed description of existing artificial neural network models. The common feed-forward
network is described first, then the training process is explained and some typical applications
are presented. The chapter continues with description of competitive neural networks. There
are several models explained including famous Self-organizing Network and some other models
such as Growing Neural Gas which will be extended and applied in further chapters.

The second chapter deals with the classification problem when it is solved by competitive
networks. A new approach to the competition driven classification is suggested [Fedorcak09].
The method is based on alternative understanding of the input data where class labels are
processed in different way. Several positive attributes such as probabilistic classification ability
are discussed and some experiments proving these improvements are also included.

The third chapter is focused on supportive abilities of selected competitive networks. In the
first part of this chapter, the weight initialization process of Radial Basis Network is inspected
and some existing approaches are examined. Next, an innovative approach is presented where the foregoing Joined Growing Neural Gas is used for RBF initialization purposes. Again, some experiments were made to support presented thoughts.

The second part of the third chapter aims at the high-dimensional data visualization problem. First, existing methods are examined and next, an innovative merge of the competitive network and the simulation of the physical system is presented [Fedorcak07]. The resulting method is used for visualization of high-dimensional datasets and appropriate experiments and figures are also included.

The last chapter goes deep into the paradigm of competitive networks. A new insight into the concept of competition is presented. Similarities between the particle/spring system and the competitive network is highlighted and an alternative model of the competitive network based on such physical system is suggested [Fedorcak10]. Selected competitive models are implemented as physically based systems and resulting improvements of the learning process are demonstrated through necessary experiments.

Finally, achieved results are summarized at the end of the thesis. Moreover, a brief introduction to the my own software tool used for presented experiments and figure generation is given in the appendix. For explicitness and clarity, common notations and properties list is also included.
Chapter 2

Artificial neural networks phenomenon

2.1 History of artificial neural networks

The study of biological neural systems dates back several hundred years. The modern era and the evolution of electronics brings up a possibility to emulate these systems. The first work on this topic is credited to neuro-physiologist Warren McCulloch and mathematician Walter Pitts [McCulloch43]. They presented the very basic mathematical model of bipolar neuron and they showed that a combination of these neurons can compute any logical function. There was not any practical use of this model but they influenced other scientists (Jon Von Neumann, Norbert Wiener).

The next major step in developing artificial neural system arrived with the book *The Organization of Behavior* written by Donald Hebb [Hebb49]. The book extended McCulloch-Pitts neuron model with the synapses as a neuron interface and it presented synapsis learning rule.

In 1958, Frank Rosenblatt, a neuro-biologist at Cornell University presented very important neuron model, the perceptron [Rosenblatt58]. This model is based on McCulloch-Pitts neuron but its parameters are not bipolar (-1,0,1) but real numbers. Frank Rosenblatt constructed the learning process and proved that this process leads to correct solution (if there is any) independently on initial configuration. The first neurocomputer *Mark I Perceptron* was developed based on this research and it was successfully used for character recognition.

Between 1959 and 1960, Bernard Wildrow and Marcian Hoff developed the *ADALINE* (ADAptive LINear Elements) and *MADALINE* (MUltiple ADAptive LINear Elements) models. This new model was supplied with new learning rule and this rule is used up to the present day [Widrow62]. Bernard Wildrow also found the first company (*Memistor Corp.*) oriented on neuro-hardware. The *ADALINE* model was used as a filter to remove echoes from telephone lines.

Some other models were developed in late ’50s but still, there were two problems in artificial neural networks research. The first problem was overestimation of artificial neural networks capabilities, which resulted in opinion that such neural networks could solve any problem and that the artificial human brain will be invented soon. Along with the moral aspects pushed by novels such as *Space Odyssey 2001* a large-scale critique of neural networks raised up and fundings for research projects were endangered. The second problem was more science-based and it was the fact that one perceptron can not solve simple logical XOR function. This can be solved by connecting more perceptrons into multi-layered network but there was not any learning algorithm which could train such network. Moreover, there was no sign that such algorithm could even exist. These arguments was summarized in book *The Perceptrons* by Minsky and Papert in aspiration to move fundings from neural network research to other artificial intelligence projects [Minsky69]. This campaign was a success and nearly for twenty years the research on neural
networks based on multilayer feed-forward model was almost stopped (especially in US).

In early ’70s, a different approach to artificial neural systems was presented. The main idea was a competition between local units [Malsburg76, Grossberg76]. Based on this research and his own research activities the most important model, self-organized network, was presented in 1982 by professor Teuvo Kohonen [Kohonen82]. Another approach was presented by John Hopfield in 1982 and 1984. He presented a model inspired by the behavior of some magnetic materials and he showed similarities between these materials and existing neural networks [Hopfield82]. John Hopfield was also known for his likable character and his series of invited lectures convinced scientists and researchers to renew interest into the research of artificial neural networks.

In 1986, Rumelhart, Hinton and Williams published an algorithm called back-propagation [Rumelhart82]. This learning algorithm was intended for training a multi-layer perceptron network and caused a real breakthrough in the research of classical neural networks. Back-propagation algorithm solved the problem of training the multi-layer network and opened the gold era for artificial neural networks. Many improvements and modifications of back-propagation algorithm were presented until present day but even most basic form of this method is highly effective and it is used frequently. A famous example of practical use of neural networks is the NETtalk system developed by Sejnowski and Rosenberg [Sejnowski87]. This system is capable to convert written text to understandable speech after a relatively short training. It clearly beat down the similar DECTalk system which was composed of several hundred rules collected by linguistic specialists during many years.

2.2 Neurophysiological inspiration

The neural network serves as unique controlling system of all living creatures producing appropriate reactions to the outer world. From a functional view, the central nervous system of almost all multicellular animals can be divided into several parts, such as receptors sensing mechanical, thermal or any other impulses. A reaction to these impulses is mostly created in cerebral cortex, the core of the central nervous system. We can distinguish a number of sensory areas where the information from receptors is processed. A voluntary action is then created in association areas and executed by effectors (muscles, glands etc.).

The most basic element of neural network is a neural cell, the neuron. Every neuron is connected to a large number of other neurons and together they create a neural network. The picture below shows schematic diagram of a neuron cell [Rougier10]:

The neuron cell core soma is surrounded by cellular extensions dendrites. Dendrites (referred also as dendritic tree) serves as input channel to the neuron. The axon is a thread-like output channel of the neuron. It can extend thousands of times the diameter of the soma in length. The axon is covered with myelin sheath which helps the transmission of nerve signals away from the soma. The axon is terminated with synapses, specialized structures with chemical membranes where neurotransmitter chemicals are released in order to transmit signals to target neurons. The signal transmitted through synapses can be excited or inhibited by proper synapse receptors in dendrites of the target neuron. Neurons are able to create new connections between them or destroy existing unused connections.

When a neuron is excited an electrical signal is transmitted through its axon thread to target neurons. If the target neuron simulation exceeds specific value called threshold the neuron is excited and the synapse receptors are sightly modified. This principle can start cascades of
neuron excitement and it is believed that this is the essence of cogitative process. It is also believed that the synapse receptors configuration encodes engrammes stored in memory.

The artificial neural network is obviously inspired by its biological parallel. The research oriented to biological neural networks brought us knowledge how these systems work on basic level, what is the structure and functionality of neural networks. However, the complexity of nervous system of higher fauna including human brain is still a big obstacle. For example, the human cerebral cortex contains about 15 billions of neurons creating a heavily connected network. Even the best non-invasive diagnostic devices are not good enough to observe particular neurons and even if we could watch the neurons apart, there are not enough computational resources to process such data.

Generally, the biological neural system behaves like any other complex system. We know basic structures and rules, we can describe complex behavior on some level, but understanding the complex behavior is beyond our capabilities. Nevertheless, we can build a mathematical model of basic elements of neural networks and simulate such model. Than we can extract useful information from the model behavior.

### 2.3 Goals of artificial neural networks

The artificial neural network is an adaptive system which changes its inner structure according to the desired task of the network. There are several goals that such network can achieve but in practice, they inherit the task of creating output data according to fetched input data. The computational power of the neural network is very strong and even basic neural network models can learn very complex non-linear input to output mappings. Precisely speaking, the computational power of the neural network is the same as the power of the Turing Machine \cite{Sontag91}. 

---

Figure 2.1: Neuron cell diagram
2.3.1 Supervised learning and unsupervised learning

Generally, there are two major approaches to how the machine learning process is handled. The first one is called supervised learning. The aim of supervised learning is deduction from training examples and there is a teacher which provide external information to the network within the meaning of that every training example – training pattern consists of input data and desired output. The task of the neural network taught by supervision of the teacher is to predict the output from the input supplied to the network. The input and desired output form a function, and therefore, the goal of the network is approximation of this function. Usually, the exact name for the goal depends on the domain of desired output. If the network operates above real numbers then the goal is called function approximation. If the network output is a finite set of values rather then a continuous interval then the goal is called classification and training data is often called labeled data, i.e. a set where every input is assigned a class label.

The second approach is unsupervised learning. Unsupervised learning does not rely on pairs of input data and desired values but rather it tries to find similarities or structures within the input dataset. The majority of unsupervised learning algorithms is based on the competitive principle and such networks are called competitive networks. The common result of unsupervised learning is a set of reference units which describe organization of the input dataset and the process is often called self-organization [Kohonen90].

Usually, for practical applications, an interpretation of reference units is needed. The interpretation is often human-aided or some heuristics method may by applied. After such interpretation we may use competitive network for classification purposes. There are some learning methods (such as Learning Vector Quantization [Kohonen92]) which utilize the competitive learning principle but are applied on labeled data. We may call them semi-supervised learning methods.

2.3.2 Other goals of artificial neural networks

There are many different models of artificial neural networks and many exotic goals can be achieved. For example, it is possible to utilize the inner structure of the trained network for different purposes then the function approximation task. As a sub-goal, the network may perform feature extraction tasks and this behavior may be utilized [Hinton06, Kerschen04]. Moreover, the structure of the network can serve as the transport channel and in such way, the compression task may be achieved [Mahoney00, Rudenko08].

2.4 Feed-forward networks

This section provides a basic insight into the most common artificial neural network – the feed-forward network. Basic model and most used learning algorithms are introduced in the latter part of the section.

2.4.1 Mathematical model of neuron cell

The basic element of the feed-forward network is perceptron [Minsky69] which is a simple approximation of biological neuron. There are several historical models such as McCulloch neuron

\[1\] Detailed descriptions of some emphasized terms are listed in the appendix chapter at the end of the thesis.
model [McCulloch43] but the perceptron model is probably the most usable one. The following figure (fig. 2.2) shows the formal structure of the perceptron.

![Perceptron Diagram](image)

Figure 2.2: Perceptron diagram

The perceptron can accept any number of input signals \( \{x_1, \ldots, x_n\} \in \mathbb{R} \). Input signals are scaled with synaptic weights \( \{w_1, \ldots, w_n\} \in \mathbb{R} \). The sum of weighted inputs forms an inner potential \( z \):

\[
z = \sum_{i=1}^{n} w_i x_i
\]  

(2.1)

According to the definition of the inner potential (eq. 2.1), negative values of the synaptic weight simulates an inhibiting synapse and positive values of the synaptic weight simulates an exciting synapse.

The output of the perceptron is defined by its output signal \( y \) which depends on the inner potential and the threshold \( \theta \in \mathbb{R} \). This dependency is defined by transfer function \( \varphi(z) \). There are several transfer function used within perceptron model. The most basic transfer function is the step function:

\[
\varphi(z) = \begin{cases} 
0 & \text{if} \quad z \geq \theta \\
1 & \text{if} \quad z < \theta
\end{cases}
\]  

(2.2)

The evaluation of the perceptron output value is synchronous. First, input signals are provided. Next, the inner potential is computed. Finally, the inner potential is transferred to the output value. This process may be referred as firing the perceptron.

Single perceptron is able to divide provided vectors of input signals into two groups by its output value. Mathematical representation of a plane clearly explains this ability. Equations for plane in \( E^2 \) and hyperplane in \( E^n \) are:

\[
n_1 x_1 + n_2 x_2 + c = 0
\]  

(2.3)

\[
\sum_{i=1}^{n} n_i x_i + c = 0
\]  

(2.4)

where \( n = (n_1, \ldots, n_n) \) is a normal vector of the plane/hyperplane and \( c \) is a real constant. Input signals are presented as n-dimensional input vector \( x = (x_1, \ldots, x_n) \in E^n \). Obviously, the hyperplane (eq. 2.5) divides input space \( E^n \) of the perceptron into two subspaces where input
vectors from one subspace generate output value 0 (the perceptron is \textit{passive/unexcited}) and input vectors from the other subspace generate output value 1 (the perceptrons is \textit{excited}).

\[ \sum_{i=1}^{n} w_i x_i - \theta = 0 \]  

(2.5)

The dividing ability of the perceptron is often denoted as the \textit{linear separation} ability. More formally, it can be described as follows: let there be an \textit{input set} of n-dimensional vectors where every vector is classified with a single binary value. Let the vectors be distributed into two groups. According to the distribution of input vectors, there is (fig. 2.3a) or there is not (fig. 2.3b) a configuration of perceptron’s weights and threshold where the perceptron "recognizes" defined groups; i.e. the output of the neuron is equal to desired value of every input signals vector.

![Perceptron separation](image)

Figure 2.3: Perceptron separation

Finding such configuration is often called \textit{perceptron learning} and evaluated input set is referred as \textit{training set}. Some learning methods will be discussed later.

2.4.2 Network of perceptrons

The example from previous section shows an input set where the \textit{linear separation} is not possible. However, a well-structured network of perceptrons can solve this task. There are many methods how such network my be structured. The most common structure is the \textit{fully-connected} feed-forward network which is also called as the \textit{Multi Layer Perceptron} (MLP) model. In this model, neurons are organized into several layers and every neuron output signal is passed to every neuron in the next layer as one of its input signal. Usually, there is an input layer of simple units which only pass input signals to the next layer. Then a number of hidden layers may be placed. The last layer is called the output layer and provides output values of the network. The vector \( \mathbf{x} = (x_1, \ldots, x_n) \) denotes the \textit{network input} and the vector \( \mathbf{y} = (y_1, \ldots, y_m) \) denotes the \textit{network output} (fig. 2.4).

The output of the MLP network is usually evaluated synchronously. First, an input vector is fetched to the input layer. Then the first hidden layer is fired, i.e. every perceptron in the layer is fired. The firing process may run in parallel because there are no lateral connections.
between perceptrons within the same layer. Output values of fired perceptrons are forwarded to the next layer and the evaluation process continues through remaining layers. Finally, the output vector is assembled from output values of perceptrons in the last layer of the network.

The next example will explain the recognition power of the MLP network. Let there be a two-dimensional input set of evaluated input vectors (fig. 2.5a). Let there be a MLP network created by one hidden layer with three perceptrons and an output layer with two perceptrons (fig. 2.5b).

There are three different classes defined in the input set, and therefore, there is a need of two outputs $y = (y_{p4}, y_{p5})$ to distinguish these classes (0,0) – black points, (1,0) – crossed points, (0,1) – white points).

Practically, three outputs are often used for such task where the one-of-C coding technique defined in latter chapters of the thesis is used) but for presentation purposes, there can be only two of them.
The position of hyperplanes constructed from perceptrons configurations is visualized below (fig. 2.6). The input space of the first layer is identical to the input space of the network. Perceptrons will construct a set of lines dividing the input space into several areas (fig. 2.6a). Every area may be marked with a triple of perceptron output values \((y_{p1}, y_{p2}, y_{p3})\), e.g. \(A\) is \((1, 1, 1)\) or \(F\) is \((1, 0, 1)\). The second layer (the output layer) has two neurons with three inputs, therefore the input space of the second layer is three-dimensional. The step transfer function (eq. 2.2) used by perceptrons causes that every point from the network’s input space is projected to one of the corners of a unit cube in the second layer input space. The corners of the unit cube is marked with appropriate letter according to the area in the input space (fig. 2.6b).

Perceptrons in the second layer will construct planes dividing their input space into several subspaces. Proper configuration of perceptron parameters cuts out interesting corners and realizes target function. The perceptron \(p_4\) will be excited for every input signal projected to area \(C\). The perceptron \(p_5\) will be excited for every input signal projected to the union of \(A \cup B\).

The previous example showed that the MLP network with one hidden layer can bound convex areas. However, more complicated non-convex cases need to be covered. The solution lies in addition of another layer to the network. The MLP network with two hidden layers can bound all possible areas. Generally, it is correct to say that the first layer will divide the input space into desired subspaces, the second layer will realize \(AND\) function above them forming convex areas and the output layer will realize \(OR\) function which chooses correct areas. One can not forget that the network may have several outputs. The network is fully-connected, and consequently, areas bound by particular perceptrons might be combined together.

### 2.4.3 Sigmoid transfer function

The transfer function of the perceptron is inspired by a real neuron cell behavior. The biological neuron will excite if its inner chemical potential get over some threshold value (by opening specific neuro-transmitter gates which will cause cascade of chemical reactions). The step transfer function has been already described above (eq. 2.2) and it has been shown how such function
affects the artificial model of the neural network. However, the mathematical model of the neuron cell could use different transfer functions to describe the potential-to-threshold behavior. One of the main disadvantages of the step function is that it is constant on whole domain and it is not differentiable in the point of threshold value. For these and some other reasons that will be addressed later, the sigmoid function needs to be introduced:

\[ y = \frac{1}{1 + e^{-\lambda (z - \theta)}} ; \quad \lambda \in \mathbb{R} \]  

(2.6)

The sigmoid function is non-linear, continuous and differentiable on whole domain. The \( \lambda \) parameter is affecting the steepness of the function near the threshold value. If \( \lambda = 0 \), then the function become constant \( y = 0.5 \). With high lambda values, the function will approximate the step function closely.

\[ \lambda = 0 \quad \lambda = 1 \quad \lambda = 5 \]

Figure 2.7: Sigmoid transfer function

2.4.4 Transfer functions and universal approximation

It has been proven that the feed-forward network with units equipped with specific transfer function might become an universal function approximator.

The choice of transfer function affect the result crucially. There are several conditions that have to be met to achieve the universal approximation attribute within the MLP network [Cybenko89, Hornik91]. Briefly speaking, the transfer function needs to be non-constant, bounded and continuous. Obviously, the sigmoidal function meets these conditions, hence the MLP network with sigmoidal transfer function is the universal function approximator.

There are different models with so-called “local units” that use radial basis functions as their transfer functions. These models will be described in the latter chapter (see sec. 4.1.1) but now, it is opportune to point out the universal approximation ability of such model [Park93].

Finally, there is a model which uses the transfer function constructed as a combination of local and global approach. This semi-local transfer function is a Gaussian function applied on standard weight-based potential value. Such network has several positive abilities (the learning is faster than in the sigmoidal perceptron network and does not suffer from the generalization problem of the RBF network[3]) but it does not have the attribute of universal approximation [Kurkova92].

[3]to be more specific, the RBF network can have problems with generalization if the training set contains irrelevant inputs [Sima96]
2.5 Adaptation of MLP network

The adaptation of the MLP network is a process of modifying weights values through time. The goal of adaptation/learning process is to learn particular training set. In different words, the goal is to find a configuration of synaptic weights where the error generated by the network is as low as possible.

The learning process runs on specific training set which usually contains a finite number of elements. Every element of the training set is a double of vectors \( p = ((\xi_1, \ldots, \xi_n), (d_1, \ldots, d_m)) \) called pattern. The input signal \( \xi \) is fetched as the input vector \( x \) to the network. The vector \( d \) is a vector of desired output values that the network should answer when input an \( x \) is provided. The most used method of quantization of the network’s error is a sum of squared differences between desired and real output throughout the entire training set. Such definition is quite natural and it has been found useful for adaptation purposes.

\[
E = \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{m} (y_j - d_j)^2
\]  

(2.7)

2.5.1 Backpropagation

The backpropagation is well-known algorithm for adapting weights of the feed-forward neural network. Introducing the backpropagation algorithm [Rumelhart82] was a great success and it is used (with many modifications) until present day. The main principle of the algorithm stands for propagating the network error back to every single neuron inside the network according to the fraction of error generated by this particular neuron. Therefore, it is crucial to know how exactly the network error is generated. According to the equation (eq. 2.7), the network error can be assumed as a function of network current state i.e. the configuration of weights and thresholds. Then the adaptation of the network is a process of finding global minimum of the error function \( E(w) \). Due to the non-linear nature of the error function, it is impossible to find the solution analytically but there are various numerical approaches for such task. The gradient descent method may be applied effectively and can be described as follows:

The learning process consists of finite number of training steps. Every training step has a number of substeps:

1. Choose a pattern \( k \) from the training set.
2. Fetch an input signal \( \xi \) as the input vector \( x \) to the network.
3. Fire the network.
4. Gather the output vector \( y \) and match it against the desired output \( d_k \).
5. Compute of the weight change \( \Delta w_{ij} \) for every synapse in the network in order to lower an error inflicted by this particular pattern.
6. If the mean squared error is not low enough repeat the algorithm from step 1.

\(^4\)Precisely, the number of elements in the training set can be unknown (or infinite) if elements are created dynamically during the training process, but there are some restrictions to apply if such behavior is needed.
The basic form of backpropagation algorithm provides the definition of the change $\Delta w_{ij}$ for every $i$-th weight within a single neuron $j$:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} + \mu \Delta w'_{ij} ; \quad \eta, \mu \in \mathbb{R}^+$$  \hspace{1cm} (2.8)

The learning coefficient $\eta$ will affect the speed and convergence of the adaptation. The second part of the equation denotes the momentum factor. The momentum describes the influence of the weight change from previous adaptation step $\Delta w'_{ij}$. The momentum may speed up the adaptation significantly and it is controlled by the momentum coefficient $\mu$. The final formulation of the partial derivative (eq. 2.8) reflects the back propagation principle of the algorithm:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \lambda_j y_j (1 - y_j) x_i$$  \hspace{1cm} (2.9)

It is assumed that every neuron $j$ has the sigmoid transfer function (eq. 2.6) with the steepness factor $\lambda_j$. The neuron produces the output value $y_j$ and every weight $w_{ij}$ is fed with the appropriate input $x_i$. The coefficient $\delta_j$ differs between outer and hidden layers. If a neuron belongs to the outer layer then the $\delta_j$ component depends only on the network output and the desired output (eq. 2.10). If the neuron belongs to the hidden layer then the $\delta_j$ component depends also on neurons in the layer above it (2.11). The upper indices in the following equations mean that values belong to the neuron in the upper layer. The coefficient $\delta_j$ differs between outer and hidden layers. If a neuron belongs to the outer layer then the $\delta_j$ component depends only on the network output and the desired output (eq. 2.10). If the neuron belongs to the hidden layer then the $\delta_j$ component depends also on neurons in the layer above it (2.11). The upper indices in the following equations mean that values belong to the neuron in the upper layer.

$$\delta_j = y_j - d_j$$  \hspace{1cm} (2.10)

$$\delta_j = \sum_{i=1}^{m} \delta^i \lambda^i y^i (1 - y^i) w^{ij}$$  \hspace{1cm} (2.11)

Since the equation above is clearly iterative, it is necessary to start the weight change evaluation from the outer layer and progress back through the network.

### 2.5.2 Batch learning and incremental learning

There are several applications of weight change values. Most used are incremental learning and batch learning. Incremental learning will update synapse weights immediately after the single pattern is processed by the network. The batch learning will accumulate weight changes to be applied after all patterns from the training set are processed. The usage of particular method depends on many factors. For example, the incremental learning is the only way the weight change can be applied if training patterns are generated dynamically during the training process. On the other hand, there are methods that forbid the usage of incremental learning due their natural behavior and therefore, the choice of the adaptation algorithm is closely tied to the origin of the training data and known facts about the function that has to be approximate.

---

5For example, Conjugate Gradients [Hagan96]
2.5.3 Improved backpropagation

It is a known fact that the basic backpropagation algorithm might cause serious problems with the speed of convergence and with the configuration of learning factors. The initial configuration may cause the learning process to start in a position where the error function is very flat, and consequently, the learning advances slowly and it might be useful to increase the learning factor. On the other hand, the learning factor might become very large when the training process reaches a steep area of the error function and the desired global minimum cannot be hit.

There are many approaches that solve this problem. For example, the RPROP (resilient backpropagation) is a very efficient and simple method where a kind of heuristic is applied to the choice of the learning rate [Riedmiller93]. Briefly, only the sign of the partial derivative is taken into account and the learning rate is multiplied each step if the sign of the derivative does not change from the last step. Moreover, some backtracking procedures may be used to improve the algorithm [Igen00].

Another different solution of the learning rate problem is the Quickprop algorithm [Fahlman88, Veitch08]. The method utilizes the information about the curvature of the error function stored in the second order derivative. It is assumed that the error function might be approximated by the quadratic function within the close area around the current position and attempts to move to the minimum of the parabola in one learning step.

2.5.4 Other methods

Generally, it is possible to use many different methods to adapt the MLP network because the task can be transformed to the problem of function minimization (or global optimization). One of the most popular methods is the evolutional optimization. There are many algorithms that belong to the family of evolutional approaches; for example genetic algorithms, differential evolution etc. The idea of evolution of the network state is very simple and straightforward. Practically, two steps are needed when the evolutional approach is applied. First, the phenotype to genotype transformation needs to be designed. Next, the cost function has to be defined. Both of attributes has been proposed for the task of adaptation of the MLP networks [Ilonen03, Belew90]. Various approaches has been published so far, including the evolution of the network structure [Chen06, Gruau94].
2.6 Competitive networks

There is a large set of artificial neural networks called competitive networks. These networks are rather different from common feed-forward networks. The main difference is the way the competitive network learns an input dataset. While the common MLP network learns to produce the desired output according to information provided by a teacher (*supervised learning*), the competitive network learns the structure of input data through the competition between its own neurons (*unsupervised learning*). Due to the different kind of learning process, applications of competitive networks are different. The common tasks are data clustering tasks or vector quantization task. However, there are some methods where the teacher is present and the competitive principle can also be used for supervised classification.

2.6.1 Common model of competitive network

The structure of a common competitive network is very simple. Usually, it is a set of neurons organized in a single layer. Every neuron is connected to all inputs and there is a weight associated with every input. Sometimes, lateral connections between neurons are present and these connections may affect the learning process significantly. There is no potential, no transfer function and no output as has been introduced in the feed-forward model. Also, weights have a slightly different meaning from weights in the MLP network.

It is not a surprise that there is a biological inspiration for competitive networks. It is called *lateral inhibition* [Bekesy67, Lachman79]. The inspiration was taken from special kinds of neural structures (occurring, for example, in sensory networks) where excitation of a neural cell leads to inhibition of neighboring neural cells. Such behavior may be understood as the competition of neural cells, and the artificial model follows this idea.

![Lateral inhibition in auditory neural network](image)

The adaptation of a competitive network is a process of learning the distribution of the input dataset through the competition between neurons. Neurons may be understood as reference vectors with the same dimensionality as input data (there is a weight associated with every input). The competition is metric driven. Usually, the *Euclidean metric* is used but there are some other possibilities discussed below. The learning process consists of a fixed number of repetitive steps. Every step contains several substeps:

1. Determination of the winner according to the metric
2. Adaptation of the winner using some adaptation rule
3. Optionally, if some sort of topology is defined, the neighborhood of the winner may be adapted as well.

Most of the competitive learning methods use a very similar learning step \cite{Fritzke97}. Sometimes, the network is able to change its structure dynamically by adding or removing neurons — growing networks; some other attributes such as a local accumulation of distortion error may be used for learning purposes. Nevertheless, it is necessary to set initial positions of neurons before the learning process is started and there are several ways in which such a task can be done (e.g. random distribution).

### 2.6.2 Hard Competitive Learning

Hard competitive learning is the most basic model of the competitive network. The learning process is based on the search of the closest unit (the winner) and only the winner is adapted; therefore, such method is often called winner-take-all learning.

Various results may be obtained by different ways of processing adaptation changes. Input signals may be processed incrementally — the winner is adapted after each step (K-means \cite{Kanungo02}) or the adaptation can be done in a batch where neurons are adapted after all input signals from the whole dataset are processed (LBG method \cite{Linde80}). A simplified algorithm of the incremental learning is described below.

1. Initialize the set \( N = \{ u_1, u_2, ..., u_m \} \) of \( m \) neurons randomly according to the dataset distribution where every neuron \( u_i \) has its own weight vector \( w_{u_i} \in \mathbb{R}^n \).
2. Generate an input signal \( \xi \) at random from the dataset.
3. Determinate the winner \( w = w(\xi) \) of the competition:
   \[
   w(\xi) = \arg \min_{u \in N} \| \xi - w_u \| 
   \] (2.12)
4. Adapt the reference vector \( w_w \) of the winner \( w \) by moving it towards the input signal \( \xi \):
   \[
   \Delta w_w = \eta(\xi - w_w) ; \quad \eta \in \mathbb{R}^+ 
   \] (2.13)
5. Until the condition for ending is reached (e.g. maximum of learning steps)
   continue with step 2.

The initialization of positions of neurons is a crucial step for hard competitive methods. If neurons are not well-distributed (e.g. simple random distribution is used) then it is likely that there will be ”dead units” present in the final configuration of the network. These units never won the competition, and therefore, they kept their initial position for the whole learning process. Such neurons are completely useless and must be avoided. A common way to avoid dead units is to use randomly chosen input signals from the input dataset to initialize neuron positions. Additionally, positions may be noised within a meaningful interval.

\*\*See sec. 2.5.2 for similar approach in the feed-forward network adaptation process.\*\*
However, there is another problem even if the initialization of positions is driven by the dataset distribution. For example, if the goal is error minimization (a sub goal for the classification task) and the distribution of the input dataset is highly non-uniform, then the final configuration of the network may be unbalanced, i.e. there may be regions where neurons are concentrated, or by contrast, some regions may be undersampled. The following figures (fig. 2.9) show these problems on the simple two dimensional dataset borrowed from the literature [Fritzke97].

![Figure 2.9: Hard competitive learning – two different final configurations. Undersampled ($\alpha$) and oversampled cluster ($\beta$) can be observed. The left picture shows the case with dead units that occurred due to the simple random initialization. The position of every neuron was recorded during the learning process. These records form trajectories that are displayed as red lines and may help to understand the progress of the competitive learning.](image)

2.6.3 Soft competitive networks without fixed dimensionality

There are several techniques that have been developed to avoid situations described above, for example growing structures or winner-takes-most approaches. In this section, several methods from the area of soft competitive learning will be presented.

There are two major differences between ongoing models and the model described in the previous section. The first difference is the soft competition which means that not only the winner is adapted but other units (e.g. neighbors) may be adapted as well. The second difference is that the algorithm may produce some kind of topology during the learning process in the manner of connections between units. However, the topology has no fixed dimensionality.

**Neural Gas**

The Neural Gas algorithm, as a representative of soft competitive methods, divides the adaptation between all neurons according to the distance from the input signal [Martinetz91]. The algorithm can be described in several steps [Fritzke97]:

1. Initialize the set $\mathcal{N} = \{u_1, u_2, ..., u_m\}$ of $m$ neurons randomly according to the dataset distribution where every neuron $u_i$ has its own weight vector $w_{u_i} \in \mathbb{R}^n$.

2. Initialize the time parameter $t = 0$;
3. Generate an input signal $\xi$ at random from the dataset.

4. Order all elements of $N$ according to their distance to the input signal $\xi$, i.e. find the sequence of indices $(k_0, k_1, \ldots, k_m)$ where $\|\xi - w_{k_0}\| < \|\xi - w_{k_1}\| < \ldots < \|\xi - w_{k_m}\|$. The rank number (the index) associated with every neuron $i$ is denoted by $k_i(\xi, N)$.

5. Adapt all reference vectors using this adaptation rule:

$$\Delta w_i = \eta(t) \cdot h(\lambda(t), k_i(\xi, N)) \cdot (\xi - w_i)$$

(2.14)

where functions $\eta(t)$ and $h(\lambda(t), k)$ realize time and distance decaying factors:

$$\eta(t) = \eta_i \left( \frac{\eta_f}{\eta_i} \right)^{t/t_{max}}, \quad \lambda(t) = \frac{\lambda_f}{\lambda_i} \left( \frac{\lambda_f}{\lambda_i} \right)^{t/t_{max}}$$

(2.15)

$$h(\lambda(t), k) = \exp \left( - \frac{k}{\lambda(t)} \right)$$

(2.16)

6. Increase the time parameter, $t = t + 1$. If $t < t_{max}$ continue with step 2.

Suitable values for $\eta_i, \eta_f, \lambda_i, \lambda_f \in \mathbb{R}^+$ have to be chosen. Pictures below (fig. 2.10) present three stages of the learning process for the same two dimensional dataset which has been presented above.

Figure 2.10: Neural Gas algorithm, two-dimensional example

**Growing Neural Gas**

The *Growing Neural Gas* algorithm [Fritzke97] is derived from the *Neural Gas* algorithm with one main difference which is allowing the network to grow. Neurons become inter-connected during the learning process and a graph structure is generated. The graph is treated as a neighborhood and it is utilized within the learning process. A detailed description of the GNG algorithm is given below:
1. Initialize the set of neurons $\mathcal{N}$ with two units $u_0, u_1$ according to the distribution of the learning dataset. Initialize the set of connections $\mathcal{C} \subset \mathcal{N} \times \mathcal{N}$ to the empty set $\mathcal{C} = \emptyset$. It is presumed that the relation $\mathcal{C}$ is reflexive and any addition or removal of a connection will also affect its opposite member.

2. Generate an input signal $\xi$ at random from the dataset.

3. Find the winner $w_0$ and the second winner $w_1$ according to the chosen metric:

$$w_0 = \arg \min_{u \in \mathcal{N}} \| \xi - w_u \|, \quad w_1 = \arg \min_{u \in \mathcal{N} \setminus \{w_0\}} \| \xi - w_u \|$$

(2.17)

4. If the connection between $w_0$ and $w_1$ does not exist, create it. Set the age of the connection to zero (even if already exists):

$$\mathcal{C} = \mathcal{C} \cup \{(w_0, w_1)\}, \quad \text{age}_{(w_0, w_1)} = 0$$

(2.18)

5. Increase the local error $E_{w_0}$ of the winner by the following value:

$$\Delta E_{w_0} = \| \xi - w_{w_0} \|$$

(2.19)

6. Adapt the weight vector of the winner and also adapt the weight vectors of winner neighbors:

$$\Delta w_{w_0} = \eta (\xi - w_{w_0}); \quad \eta \in \mathbb{R}^+$$

$$\forall u \in \mathcal{U}(w_0); \quad \Delta w_u = \mu (\xi - w_u); \quad \mu \in \mathbb{R}^+$$

where $\mathcal{U}(u) \subset \mathcal{N}$ and $u_k \in \mathcal{U}(u) \leftrightarrow (u, u_k) \in \mathcal{C}$

(2.20)

(2.21)

7. Increase the age of all connections emanating from the winner $w_0$:

$$\forall u \in \mathcal{U}(w_0); \quad \text{age}_{(w_0, u)} = \text{age}_{(w_0, u)} + 1$$

(2.22)

8. Delete all connections older than $\text{age}_{\text{max}}$.

9. If the $\lambda$ steps passed from the last network growing phase, grow the network:

(a) Find the neuron $e$ with the maximum local error. Find among neighbors of $e$ the neuron $q$ with the maximum local error:

$$e = \arg \max_{u \in \mathcal{N}} E_u, \quad q = \arg \max_{u \in \mathcal{U}(e)} E_u.$$ 

(2.23)

(b) Create a new unit $r$ by linear interpolation of units $e$ and $q$.

$$\mathcal{N} = \mathcal{N} \cup \{r\}, \quad w_r = \frac{w_e + w_q}{2}$$

(2.24)

(c) Create connections $(r, e)$ and $(r, q)$ and delete connection $(e, q)$:

$$\mathcal{C} = \mathcal{C} \cup \{(r, e), (r, q)\}, \quad \mathcal{C} = \mathcal{C} \setminus \{(e, q)\}$$

(2.25)
(d) Decrease local error of neurons $e$ and $q$ by factor $\alpha \in \mathbb{R}^+$ and interpolate error value for the new unit $r$:

$$
\Delta E_e = -\alpha E_e,
\Delta E_q = -\alpha E_q
$$

$$
E_r = \frac{1}{2}(E_e + E_q) \quad (2.26)
$$

10. Decrease local error of all neurons by factor $\beta \in \mathbb{R}^+$:

$$
\forall u \in \mathcal{N} ; \quad \Delta E_u = -\beta E_u \quad (2.27)
$$

11. If the condition for ending was not met (e.g. maximum of nodes or local error level), continue with step 2.

After several experiments, we are able to discover some basic characteristics of the GNG model. During the learning phase the network occupies given input space in a short time and then it focuses on more precise distribution by growing into a dense network. This behavior can be observed on a simple two-dimensional example (fig. 2.11).

![Figure 2.11: Growing Neural Gas algorithm, two-dimensional example](image)

(a) 5000 iter.  (b) 12000 iter.  (c) 30000 iter.

Another example shows the GNG algorithm in three dimensions. The input dataset contains 2000 samples randomly chosen from a defined part of the space. Dataset distribution can be imagined as the character H twisted along the axis X (fig. 2.12). The following pictures (fig. 2.13) present the resulting network during the learning process.

As we mentioned before, the algorithm can describe dataset distribution within a relatively short number of iterations and the structure of the network is spatially similar to the dataset distribution. After 40000 iterations, neurons are uniformly distributed through the whole dataset.

The graph structure generated by the GNG algorithm is very useful. For example, it is proved [Martinetz93] that weight vectors and edges between them are part of Delaunay triangulation. Another usage of this structure is discussed later (see sec. 4.2), where the method for visualization of high-dimensional data is presented.

2.6.4 Soft competitive networks with fixed dimensionality

In this section, several methods from the area of soft competitive learning with fixed dimensionality will be presented. The major difference from algorithms presented in the last section
is that the dimensionality $d$ of the structure generated by the network is fixed and has to be chosen in advance. There is an obvious advantage to this approach, thus such method defines mapping from $n$-dimensional space into $d$-dimensional space where $n > d$. If $d = 1, 2, 3$, then the structure of the network can be used for the visualization of the processed dataset.

**Self-organizing Feature Map**

The Self-organizing Feature Map, also known as Kohonen Map \cite{Kohonen90}, is very famous and probably the most used competitive network. There are similarities between this method and NG/GNG algorithm since the decaying adaptation factor and neighborhood influence is used. However, there is one major difference in the neighborhood definition. The neighborhood of the SOM is defined by the fixed grid structure which does not change during the learning process. The grid structure may be defined in various dimensions but in most cases, two dimensions are used. Also, the grid structure may be organized in several ways (e.g. simple square grid or hexagonal grid).

Along with the definition of the structure, the neighborhood scheme has to be defined because the adaptation strength is driven by the distance to the winner unit. The common 4-neighborhood (and appropriate Manhattan distance) or the 8-neighborhood (Chebyshev distance) may be used. If the hexagonal grid is chosen, the 6-neighborhood and hexagonal distance is used. The picture above (fig. 2.14) shows various neighborhood schemes. The algorithm of the Self-organizing Feature Map can be described as follows \cite{Fritzke97}:

1. Initialize the set of neurons $\mathcal{N} = \{u_1, u_2, \ldots, u_m\}$ with reference vectors $w_u \in \mathbb{R}^n$ at
random values according to the distribution of the dataset. Initialize set of connections \( C \subset \mathcal{N} \times \mathcal{N} \) according to the chosen grid type. The grid is fixed during the learning process.

2. Initialize the time parameter \( t = 0 \).

3. Generate an input signal \( \xi \) at random from the input dataset.

4. Determine the winner \( w = w(\xi) \) of the competition:

\[
w(\xi) = \arg \min_{u \in \mathcal{N}} \| \xi - w_u \|
\]  

(2.28)

5. Adapt all neurons in network according to the distance from the winner. The winner is also adapted; its distance is equal to zero:

\[
\forall u \in \mathcal{N} ; \quad \Delta w_u = \eta(t) \cdot h(w, u, t) \cdot (\xi - w_u)
\]  

(2.29)

where functions \( \eta(t) \) and \( h(w, c, \sigma(t)) \) realize time and distance decreasing factors:

\[
h(w, c, \sigma(t)) = \exp\left( -\frac{d(w, c)^2}{2\sigma(t)^2} \right) ; \quad d \in \{ d_\infty, d_1, d_H, \ldots \}
\]  

(2.30)

\[
\eta(t) = \eta_i \left( \frac{\eta_f}{\eta_i} \right)^{t/t_{\max}} , \quad \sigma(t) = \sigma_i \left( \frac{\sigma_f}{\sigma_i} \right)^{t/t_{\max}}
\]  

(2.31)

The figures below (fig. 2.15) show several stages of the SOM learning process applied to the same two-dimensional dataset as before. At the beginning, the network structure is highly disorganized, and therefore, neurons are distributed at random within the range of the input dataset. As the learning process advances, the network structure forms a surface rapidly and tries to concentrate on clusters of input signals. Such behavior is very useful and applicable in various ways. A typical application of the SOM algorithm lies in the area of visualization and it is given in the further chapter (see sec. 4.2).
2.6.5 Other methods

There are many other competitive methods that have been developed in recent years and it is out of the scope of this thesis to present them all. However, I would like to mention some of them for their specific and interesting attributes.

Growing Grid, Growing Cells [Fritzke93, Fritzke95a] Practically, the Growing Grid model is a SOM which grows by the addition of rows and columns. The growing phase of the algorithm counts the number of wins for every neuron and this measure is used to determine the grow point. The positive attribute of this model is the ability to maintain the two-dimensional structure; however, the addition of a whole row or column may lead to inadequate distribution of units, i.e. the presence of dead units. The Growing Cells is a very similar model to the previous model but the network is not structured as a regular grid but rather as a structure created by \( k \)-dimensional simplices. The algorithm is almost the same as the GNG algorithm; some rebuilding steps are present to maintain the \( k \)-dimensional structure.

Growing Hierarchical SOM [Dittenbach02] The principle of this model is a mixture of Growing Grid and the hierarchical tree structure. Again, local distortion error levels are used to determine the point where the map should grow. The algorithm extends the map in two ways. First, rows or columns may be added if the error level is high. Second, the network can add layers of new SOMs under problematic units.

Self-Splitting Competition [Zhang02] This model is rather different from the models presented above. The authors criticize the standard approach to the competitive network and its inability to choose the proper number of prototypes. The method aims to follow the one-prototype-takes-one-cluster paradigm and a method for how such behavior may be achieved is suggested. Moreover, a splitting criterion is presented, and consequently the network can create new units and acquire all natural clusters that are present in the dataset.
2.6.6 Transformations of input dataset

When an unsupervised learning method is used, various types of preprocessing of the input data have to be considered. Most of applications involve the Euclidean distance somewhere in their learning algorithm. Such measure is very sensitive to relative magnitudes of input variables. Imagine one input which varies within range \((0, 1 000 000)\) and another input variable from interval \((-0.1, 0.1)\). Let the clustering of the data be the aim. There could be a very clear cluster defined by the second variable, but the magnitude of the first variable definitely overpowers the second one.

Therefore, some transformations of the input data need to be performed. Generally, it is possible to transform input variables or input signal. If we imagine input data as a matrix, then we can transform columns (input variables) or rows (input signals). Both options have advantages and disadvantages; however, the first one is mostly used. The usual transformation which can solve the distance measure problem is the normalization/standardization of input variables. Sometimes the same process is called standardization because “normalization” may call to mind vector normalization. Furthermore, the process of standardization can be interpreted in various ways as well:

- **The ranges of the input variable are known.** It is possible to subtract all input vectors by the minimum of all inputs in a column and then divide them by the range of a particular variable. After such transformation, all inputs lie inside interval \((0, 1)\) and the measure problem is gone. However, such transformation is sensitive to outliers in the dataset.

- **The ranges of the input variable are unknown.** In such situation, standard deviations and average values have to be known. It is possible to subtract all input vectors by the average instead and divide inputs by the standard deviations and achieve similar results as in the first case. Such transformation is less sensitive to outliers in the data but the information about clusters may get corrupted in some cases, for example, due to the skewness of the input variable.

The normalization of input patterns can be useful as well\(^7\). Usually, such transformation is used in addition to the cosine distance due to the nature of this distance measure. However, one must be very careful with the usage of this measure so as to avoid any serious data lost. Also, there could be a problem with irrelevant input variables.

In conclusion, the problem of data normalization/standardization is not tied only to the competitive network area but to almost all machine learning methods. For example, one may consider the standardization of a dataset when the standard MLP network with backpropagation learning is used. Theoretically, there in no need to do such transformation\(^8\) but practically, it is better to include it due to the initialization of weights and thresholds of the neurons inside the network. If the neurons are treated as hyperplanes that divide the input space of the training data, it is crucial to initialize them as effectively as possible and it is helpful to have standardized input data where all input variables are in the same range.

\(^7\)Now, the real vector normalization is meant

\(^8\)A perceptron can perform such transformation naturally
2.6.7 Effect of dimension

The dimension of the input dataset can affect the learning process significantly. For example the MNIST dataset, which will be used for experiments in the following chapters, has patterns of 784 dimensions. There are fundamental differences between data space of such dimension and low-dimensional space datasets.

Probably the most important issue of any high dimensional space is the problem called curse of dimensionality. Briefly described, if the dimension of a dataset increases then the volume of the data space increases exponentially. If we consider a Cartesian grid of spacing 1/10 within the unit square, we need 100 points; if the cube in 20 dimensions is considered, we need $10^{20}$ points to achieve the same spacing.

If we switch to the area of machine learning, the situation is very similar. Usually, the neural network performs some input to output mapping, i.e. it realizes a function from the input space to the output space, and therefore, it has to analyze the input space through input signals and has to cover the input space with some “resources” (reference units, neurons, ...). It is obvious that amount of needed resources will increase along with the hyper volume of the input space which has to be covered.

There are various representations of “resources” and “covering techniques” which might be used in the area of neural networks. Some of them are less sensitive to the dimension of the input dataset. For example, the standard MLP network uses sigmoidal neurons which can divide input space into two halves regardless to the dimension, therefore such network do not suffer from this problem critically. On the other hand, nearly all unsupervised learning methods (and RBF networks as well) use a kind of reference units which have local character and have to cover the whole dataset. In such situation, high dimension can inflict serious damage to results of the learning process.

![Figure 2.16: Relative distance comparison – Several datasets has been analyzed through distances between input signals. The results are shown as box plots, one for each dataset. Every box plot is enriched with 0.1 and 0.9 percentiles (dashed lines) and with the number of dimensions. Distances were divided by distance of the most distant points in the appropriate dataset space.](image)

---

9 see chapter 3.
Moreover, there is another issue connected to the high-dimensional input spaces, the \textit{concentration of distance} \cite{Beyer99}. It has been shown that the Euclidean distance in high-dimensional spaces tends to concentrate, and as a result, all pairwise distances between points in such spaces will be very similar to each other. A simple statistical analysis may show the concentration of distance phenomenon. The picture above (fig. 2.16) displays box plots of pairwise distances between all points in various datasets where the distance concentration is clearly visible. Notice how the minimal distance (or lower quartile) becomes longer with higher dimension of the dataset. The most distinct is the MNIST dataset where 90% of input signals are situated in positions which are in very similar distance to any other input signal.

Many unsupervised learning algorithms use Euclidean distance measure for the search of the nearest unit - the winner of the competition (see sec. 2.6). It is obvious that if the distance function concentrate with the increase of the dataset dimension, the effectiveness of the winner search decreases. In such situation, an unsupervised method will probably have problems with identifying clusters within the dataset due to the fact that all inputs are in relatively same distance.

\[
\|\xi_i - \xi_j\| \sqrt{n}
\]

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.17.png}
\caption{Relative distance comparison 2 – Class wise comparison. Two datasets has been analyzed through distances between input signals. The results were divided into two sub groups according to the class label stored in each pattern; equal labels – \textit{A}, unequal labels – \textit{B}.}
\end{figure}

On the other hand, the learning process is not useless even if the concentration of distance is present. One of the goals of unsupervised learning methods is to organize units in positions where similar reference units are related, i.e. lie near each other or are connected together. If we analyze the dataset with respect to the hypothetical goal of identifying inputs of the same class\textsuperscript{10} (see fig. 2.17), it is clear that distances between inputs with corresponding classes are shorter at average, and therefore, the method is able to utilize the winner search driven by the nearest neighbor approach.

Some solution of this unfortunate behavior of the Euclidean distance measure has been proposed, for example, the usage of other fractional norms \cite{Aggarwal01}, but still, it is not clear if this solution is applicable in general \cite{Francois07}. Additionally, it has been shown that the concentration issue is rather connected to the data itself then to the number of dimensions \cite{Francois07, Durrant09}. Nevertheless, the problem exists and it is important to remind its effect when the competitive learning method is applied.

\textsuperscript{10}which becomes very easy if we actually have datasets with such information for teaching purposes.
2.7 Other artificial neural networks

There are plenty of different artificial neural network models and plenty of methods for teaching these networks. It is almost impossible to mention all of them because the topic is moving forward very fast and there are always new approaches or improvements of existing models published every year. Also, there are models which are very similar to each other (e.g. Support Vector Machines and MLP Network) and sometimes, old methods are reinvented with a different name.

It is out of the scope of this thesis to give a full list of all possible neural network models. Only a brief description of the most used models has been given here to introduce these which will be extended or improved in next chapters. Nevertheless, some other models will be introduced later as well, usually at the point where a direct comparison with a discussed topic is needed.
Chapter 3

Self organization of labeled data

The following chapter is focused on competitive networks taught by supervision. An alternative to common supervised competitive learning is suggested and it is explained how labeled data may be processed within the standard metrics-based competitive learning.

There are some learning methods (such as Learning Vector Quantization) which can utilize the competitive learning principle but are applied to labeled data. They are called supervised competitive learning methods and they have been mentioned before as semi-supervised learning. The main difference between unsupervised competitive learning methods and supervised competitive learning methods is the structure of the input dataset. The supervision above the learning process must be based on some information added to the system. Usually, this information is a class label assigned to each input signal. The learning process is supervised through these labels and the common result of supervised competitive learning is a classifier trained on given patterns i.e pairs of input signals and their labels.

Later on this chapter, a new method of semi-supervised learning is presented [Fedorcak09]. My research has been inspired by some very interesting ideas about learning from context. The literature suggests that in some cases it is vital to focus on understanding the input dataset and leave the classification problem until later [Hinton06b]. This leads me to the idea that the pair of input data and its label should be taken as an indivisible object. The joined information may be passed to the network rather than using the label for the output correction, which is the common approach. If this idea is applied to competitive learning, useful results can be obtained.

However, it is important to understand how the standard supervised competitive learning method works. The following sections deal with this topic.

3.1 Supervised competitive learning

There are two supervised competitive learning methods presented in the following sections. The first is the very common Learning Vector Quantization method and the second one is an improvement on the Growing Neural Gas which can handle supervised learning.

3.1.1 Learning Vector Quantization networks

Learning Vector Quantization [Kohonen92] is a prototype-based classification algorithm. First a vector quantization of a given dataset has to be found. The idea of vector quantization is very simple. It is finding a set of representatives or code-book vectors that provide a good approximation of the original input dataset. Almost all competitive networks work with neurons with weight vectors distributed in the input dataset. In fact, if we think of reference vectors as code-book vectors, then all competitive networks perform vector quantization. The aim of the LVQ method is to correct the competitive learning process, so that it gives better results according to the label information stored in the training set.
There are several somewhat different LVQ algorithms appearing in the literature [Kohonen92] but they are all based on similar rules. The structure of the LVQ network is created by two layers. The first layer is a competitive layer and can even have, for example, SOM topology. The second layer is the output layer with a number of neurons equal to the number of classes in the training set. The output layer performs mapping from competitive units to classes defined in the training set.

Generally, there are two stages of the LVQ learning process. The first stage is pure competitive learning where the competitive layer is involved. The result of the first stage is neuron position (i.e. values of their weights vectors). These neurons will describe clusters in the input space where input signals are situated. The second stage sets the weights of neurons in the output layer to perform desired class mapping. Also, positions of competitive neurons could be changed within this stage if new positions suit the classification better. General steps of the LVQ algorithm are given below:

1. Initialize a set of competitive units \( \mathcal{N} = \{ u_1, u_2, \ldots, u_m \} \) with reference vectors \( \mathbf{w}_u \in \mathbb{R}^n \) at random according to the distribution of the dataset. Also, initialize connections between units if a model with connections is involved.

2. For every unit \( u \), initialize the vector \( \mathbf{l}_u = (l_1, l_2, \ldots, l_k) \) where \( k \) is the number of different classes appearing in the training set and every item \( l_i \) denotes the number of input signals mapped to this unit.

3. Perform the competitive learning (see sec. 2.6.2). In every step, increase the value of item \( l_i \) of the winner unit \( w \) according to the class assigned to the input signal \( \xi \):

\[
\mathbf{l}_w = \mathbf{l}_w + \mathbf{c}
\]

where \( \mathbf{c} = (c_1, c_2, \ldots, c_k) \) is a class vector constructed by one-of-\( C \) coding technique and it is assigned to every input signal \( \xi \), forming a pattern \( \mathbf{p} = (\xi, \mathbf{c}) \).

4. When the competitive stage is finished, assign a class value to every neuron \( u \) in the competitive layer according to the maximum element stored in \( \mathbf{l}_u \).

5. Start the second stage:

   (a) Choose a pattern \( \mathbf{p} = (\xi, \mathbf{c}) \) from the training set.

   (b) Find the winner \( w = w(\xi, \mathcal{N}) \) of the competition.

   (c) Adapt the winner by moving it towards the input signal, if the class of the input signal is equal to the most mapped class or move the winner away respectively:

\[
\Delta \mathbf{w}_w = \begin{cases} 
\eta(\xi - \mathbf{w}_w) & \text{if } \|\mathbf{l}_w\|_\infty = \mathbf{l}_w \cdot \mathbf{c} \\
-\eta(\xi - \mathbf{w}_w) & \text{if } \|\mathbf{l}_w\|_\infty \neq \mathbf{l}_w \cdot \mathbf{c}
\end{cases}
\]

   (d) Continue with step (a) until maximal iteration is reached.

The algorithm described above is denoted as the LVQ1 algorithm. There are some improved versions of this algorithm denoted as LVQ2.1 and LVQ3. For example the LVQ2.1 algorithm has
a different adaptation step, where two closest neurons need to be found and they are adapted together but only when a special condition of different classes is met (i.e. if the closest neuron belongs to the desired class and the second one does not):

\[ w_1 = w(\xi, N) ; \quad w_2 = (\xi, N \setminus \{w_1\}) \]
\[ \|l_{w_1}\|_\infty = 1 \cdot c \land \|l_{w_2}\|_\infty \neq 1 \cdot c \] (3.3)

Also, the ending condition may be altered to stop the algorithm when no adaptation was done after the whole training set has been processed.

### 3.1.2 Supervised Growing Neural Gas

The Growing Neural Gas model also has few supervised modifications, *Supervised Growing Neural Gas* - SGNG [Fritzke94]. The first SGNG model is a combination of the standard GNG model and the *Radial Basis Function Network* [Buhmann03].

The main principle of this method is to modify the standard GNG algorithm and utilize error values of output neurons to modify the position and number of RBF units. Precisely, the evaluation of the local error variable (see eq. 2.19) is modified, and as a result, new neurons are rather created in positions where different classes are present.

There is another supervised modification of the GNG algorithm [Jirayusakul07]. In this case, the model is pure self-organizing network and the supervision is driven by very similar principle to the LVQ2 algorithm (see section above). Moreover, the algorithm starts with \( k \) neurons where \( k \) is equal to the number of classes in the training set.

### 3.2 Joined Growing Neural Gas

This section is focused on the classification task performed by the modified GNG model. This modification utilizes joined information of the input signals and their labels and finally, a new classifier is presented.

#### 3.2.1 The idea of joined reference vector

As it was mentioned in the section above, there are competitive models which can use labeled data for training and performing the classification task. Most models use labels associated with input signals for correction of the network output. My approach is different. The label may be understood as a part of input data. The learning algorithm can remain exactly the same and the network organizes itself as usual. If the one-of-\( C \) coding is used for class labels, the self-organization has some interesting properties and can be used for classification directly.

It is possible to choose any competitive network for self-organization of input data created by joined information of input signal and class attributes. I have chosen the *Growing Neural Gas* method for its good abilities in self-organization. The algorithm of learning the joined dataset can be described as follows:
1. Create a set \( \mathcal{J} = \{ \mathbf{j}_1, \mathbf{j}_2, \ldots, \mathbf{j}_P \} \). Every vector \( \mathbf{j}_i \) contains joined information of input signal \( \xi \) and class vector \( \mathbf{c} \) imported from each training pattern \( \mathbf{p} \in \mathcal{T} \). \[
\mathbf{p} = ((\xi_1, \xi_2, \ldots, \xi_n), (c_1, c_2, \ldots, c_k)) \rightarrow \mathbf{j} = (\xi_1, \xi_2, \ldots, \xi_n, c_1, c_2, \ldots, c_k) \quad (3.4)
\]

2. Initialize the GNG method. Use reference vectors \( \mathbf{w}_u \in \mathbb{R}^{n+k} \).

3. Perform the standard GNG learning algorithm with a suitable learning factor etc.

When the learning phase is finished, the network can perform the classification task almost directly. Because the network has reference vectors of different dimensionality \((n + k)\) than inputs signals \((n)\), some transformations need to be done. This transformation will be discussed later (see section 3.2.3). First, the effect of the joined input space will be shown.

### 3.2.2 Properties of joined input space

The learning algorithm of the GNG model builds a graph of connected neurons. The structure of this graph depends on training data distribution.

When the input dataset is extended by class attributes, the distribution gets changed. If two inputs are assigned to the same class, the distance between them will not change because class vectors are equal. If two inputs have different class attributes, the distance between them will be enlarged by \( \sqrt{2} \) because the one-of-C coded class vectors will differ on exactly two places.

The normalization of input data is crucial at this point (see section 2.6.6). If data is not normalized, then the importance of added distance may be significantly decreased. Unfortunately, that original dimension of the input vector will also affect the significance of added distance. If we take an arbitrary normalized input dataset of dimension \( n \), then the maximum distance between any two input signals is \( \sqrt{n} \). For example, if we take a 200-dimensional dataset, then maximum distance is ten times bigger than distance between two distinct classes.

However, experiments presented below show that the added distance is enough even for high-dimensional input data if the normalization is used. The figures above (fig. 3.1) show the difference between the standard GNG model trained with unlabeled patterns and the GNG model trained with joined dataset of inputs and labels (the color of the input signal). The difference is clearly visible. The added distance is enough to limit connections between reference vectors with different class attributes.

### 3.2.3 Building a classifier – Applying restricted queries

Usually, when the classification task is performed by an unsupervised competitive network, the resulting reference vectors need to be interpreted i.e. every neuron is labeled manually. After the interpretation is done, the classification runs smoothly. The testing input signal is passed to the network and the winner neuron (the closest one) is found. The testing input signal is then classified according to the class label of the winner neuron.

The prediction phase of my classifier is different because the network was trained with joined training patterns. There are more network inputs than the number of elements in the testing input signal. We may think of an Auto-associative Memory model [Hopfield82b], where

\[^{1}\text{Naturally, supervised competitive models such as LVQ and current model do the labeling of neurons automatically.}\]
Figure 3.1: The difference between the standard GNG model and the GNG model trained with a joined dataset: The training set contains two-dimensional points distributed as concentric spirals. There are three different classes in the training set denoted by colors. The standard model has two-dimensional reference vectors; the extended model has five-dimensional reference vectors (2 spatial attributes + 3 class attributes).

The incomplete information is passed to the network and the network will reconstruct missing values. I used the same idea in my classifier. First, all reference vectors have to be projected from joined space back to the input space. The projection is trivial; all class attributes of every reference vector $w_u$ will be ignored. Formally, this is done by multiplying reference vectors by matrix $P$:

$$\forall u \in \mathcal{N} : w'_u = w_u \cdot P \quad (3.5)$$

$$P_{n+k,n} = \begin{pmatrix} I_{n,n} \\ 0_{k,n} \end{pmatrix} \quad (3.6)$$

where $I_{n,n}$ is an identity matrix and $0_{k,n}$ is a zero matrix.

Now, we can use a common strategy to find the winner $w'_w$ between the restricted reference vectors:

$$w'_w = \arg \min_{u \in \mathcal{N}} \| \xi - w'_u \| \quad (3.7)$$

The output of the network in the predictive phase is constructed from the original reference vector of the winner i.e. the output vector $y = (y_1, ..., y_k)$ is filled with class attributes of $w_w$:

$$y = w_w \cdot L \quad (3.8)$$

$$L_{n+k,n} = \begin{pmatrix} 0_{n,n} \\ I_{k,n} \end{pmatrix} \quad (3.9)$$

In this way, it is possible to build a classifier from almost any competitive model. In the following text, the GNG model extension will be called joined GNG or JGNG model [Fedorcak09].
3.2.4 Class probability prediction

As it was presented in the previous section, resulting neuron positions produced by the JGNG algorithm trained on the joined dataset \([3.4]\) can be used for predictions. Due to the nature of the output vector (actually, it is a trimmed weight vector), the classifier produces output values greater than 0 and lesser than 1, i.e. the classifier makes soft decisions. This is highly demanded and classifiers manifesting such behavior are generally considered better. However, there is another request for soft-decisioning classifiers and it is called probabilistic classification.

The idea of probabilistic classification is very simple and it means that the classifier results can be understood as a posterior probability estimate for predicted classes. It is obvious that there is a condition for the output of such classifier; the sum of output values has to be 1. This condition can be met by several methods (for example MLP networks may use the soft-max function as the transfer function for output units \([\text{Bridle90}]\)) but the JGNG classifier does not need any output transformation at all.

The rest of this section is focused on proving the fact that the JGNG classifier makes probabilistic classifications.

\[
\begin{align*}
X & \rightarrow W_1, W_2, W_3, W_4 \\
& \rightarrow (1, w_2, 1, 0), (0, 0.75, 0.6, 0, 1), (0, 0.15, 0.25, 1, 0) \\
& \rightarrow (1, w_2, 0, 1), (0, 0.75, 0.6, 0, 1), (0, 0.15, 0.25, 1, 0)
\end{align*}
\]

Figure 3.2: Class probability – low-dimensional example.

**Low-dimensional insight into joined weight space**

I would like to give some simple insight into the problem with a low dimensional example.

Let there be a two dimensional dataset created by several points where every point (input signal) has been labeled with a class flag and let there be only two classes used for labels. This gives us a 4-dimensional joined dataset (fig. 3.2) which may be viewed with a few projections. Input signals form three clusters in the original input space (fig. 3.2 left). Notice the ill-pattern in one of the clusters. This pattern will cause an uncertainty in the classification process.

With the omission of the second element, the joined dataset may be viewed in 3D (fig. 3.2 right). Due to the one-of-C coding, inputs are situated within vertical lines.

Let the GNG algorithm be run on this dataset. The final configuration of neurons may be plotted into the weight space (fig. 3.2 white circles). The 3D figure shows us the neuron \(c\) which produces soft-classification.
As it was explained above, the sum of all class variables within the weight vector \( \mathbf{w}_c \) has to be 1. Geometrically speaking, it has to be positioned on the plane \( \rho : w_3 + w_4 = 1 \). Unfortunately, the random initialization of the GNG algorithm does not guarantee such position. But, if one carefully analyzes the GNG algorithm, it becomes clear that whatever the initial positions of neurons were, all final positions will be on the plane \( \rho \) when the learning process is finished.

**Arbitrary dimension**

The condition for probabilistic classification is \( \| \mathbf{y} \|_1 = 1 \) where \( \mathbf{y} \in \mathbb{R}^k \) and \( \mathbf{y} = (w_{n+1}, w_{n+2}, \ldots, w_{n+k}) \) (eq. 3.8) i.e. the output must lie on the hyperplane \( \rho \):

\[
\rho : w_{n+1} + w_{n+2} + \ldots + w_{n+k} = 1 \quad (3.10)
\]

The initial position of any neuron is given by its weight vector, which is initialized at random. In general, the output vector lies on \( \rho \) or is inside the subspace \( X \) or \( X' \):

\[
X : w_{n+1} + w_{n+2} + \ldots + w_{n+k} > 1 \quad (3.11)
\]
\[
X' : w_{n+1} + w_{n+2} + \ldots + w_{n+k} < 1 \quad (3.12)
\]

The adaptation of the weight vector of the winner is defined as (see eq. 2.20 and 3.4):

\[
\Delta \mathbf{w} = \eta (\mathbf{j} - \mathbf{w})
\]
\[
\mathbf{w} = \mathbf{j} - \frac{\Delta \mathbf{w}}{\eta} \quad (3.13)
\]

where \( \mathbf{j} \) is a random pattern chosen from the training set and the learning factor is \( 0 < \eta < 1 \). If any neuron is inside \( X \), the following inequation is fulfilled:

\[
\left( j_{n+1} - \frac{\Delta w_{n+1}}{\eta} \right) + \ldots + \left( j_{n+k} - \frac{\Delta w_{n+k}}{\eta} \right) > 1
\]
\[
\frac{j_{n+1} + \ldots + j_{n+k} - \Delta w_{n+1} + \ldots + \Delta w_{n+k}}{\eta} > 1 \quad (3.14)
\]

With the help of one-of-C coding \( (j_{n+1} + j_{n+2} + \ldots + j_{n+k} = 1 \) for any training pattern) we may continue to:

\[
\frac{1}{\eta} \sum_{i=n+1}^{n+k} \Delta w_i < 0 \quad (3.15)
\]

We are almost finished because vector \( \Delta \mathbf{w}^{out} = (\Delta w_{n+1}, \ldots, \Delta w_{n+k}) \) denotes the change of the neuron’s position in the subspace from which the output vector \( \mathbf{y} \) is generated (eq. 3.8). In other words, the inequation above (eq. 3.15) tells us that the scalar product of \( \Delta \mathbf{w}^{out} \) and the vector \( \mathbf{n} = (1, 1, \ldots, 1) \in \mathbb{R}^k \) is negative \( (\eta \in \mathbb{R}^+) \). Obviously, the vector \( \mathbf{n} \) is the normal vector of the hyperplane \( \rho \). The negative scalar product and the fact that \( \mathbf{w} \in X \) means that the vector \( \Delta \mathbf{w}^{out} \) is pointing towards the hyperplane \( \rho \). If we change the position of the weight vector \( \mathbf{w} \in X' \) (3.12) i.e. the neuron lies “under” the hyperplane \( \rho \), the scalar product \( \Delta \mathbf{w}^{out} \cdot \mathbf{n} \) become positive. Again, this means that the change vector is pointing towards hyperplane \( \rho \).
We showed that for any valid training pattern and any initial position of neurons, all neurons will converge to positions which meets the condition for probabilistic classification. Of course, there some other conditions (e.g. a reasonable number of iterations of the GNG algorithm or a suitable value of learning factor $\eta$) but in practice, these can be easily achieved.

### 3.3 Experiments

The following section present some experiments processed with the JGNG method as they are compared to the standard LVQ method.

#### 3.3.1 Concentric spirals dataset

The first experiment has been run on the concentric spirals dataset already presented above. The advantage of this dataset is that it is possible to visualize the result of classification in two dimensions. Results are shown in the pictures above (fig. 3.3).

![LVQ classifier](a)  ![JGNG classifier](b)

Figure 3.3: The difference between the standard LVQ classifier and the JGNG classifier. The training set contains two-dimensional points (squares) distributed as concentric spirals. There are three different classes in the training set denoted by color. Colored areas show predicted class values of an arbitrary point within the input space.

#### 3.3.2 MNIST database

The dataset I used for another experiment is the popular MNIST hand-written digit database [MNIST98]. The database contains 60,000 grayscale images. Every image is a 28x28 square grid with the picture of a hand-written digit. All digits are centered and size-normalized. Every image is labeled with the number of a digit written on the image. There was no preprocessing in my experiment. By contrast, the real classifier will use some features extracted from raw pixel data, but I want to test my method and find out if it can be applied to high-dimensional non-transformed input space. Again, the LVQ classifier was run on the same dataset for comparison purposes. Results are shown in the following table (tb. 3.1):
Table 3.1: Experimental Results: Experiments were performed on the training part of the MNIST dataset. The training sequences were run 50 times for every configuration to lower the randomness of the initial neuron distribution. The JGNG model configuration was tuned to have the same final number of neurons as the LVQ method. Error values were computed from cross testing where 90% of patterns are used for the training process and the rest are used for testing purposes. Mean error value donates how many test patterns have been incorrectly classified.

Results show that our classifier is slightly better than the standard LVQ method. The improvement is not as significant as in the spiral dataset experiment but there are some other interesting attributes. The most important is the difference between the form of an answer by each classifier. While the LVQ method does the “hard” classification of the given input, the JGNG model performs soft classification. The following pictures show such behavior in some uncertain cases:

![Uncertain Cases](image)

Figure 3.4: Uncertain Cases; The reference vector position in the extended input space will produce soft answers (class probability) where the highest value may denote the class prediction. Low values within class vectors were omitted.

### 3.3.3 Intrusion detection dataset

The next experiment was run on intrusion detection dataset. The original dataset was prepared by the 1998 DARPA Intrusion Detection Evaluation Program by MIT Lincoln Labs [DARPA98]. The dataset contains 24 attack types that could be classified into four main categories, namely Denial of Service (DOS), Remote to User (R2L), User to Root (U2R) and Probing. The original data contains 744 MB data with 4,940,000 records. The dataset contains 41 attributes for each record plus one class label. There are various features in every record like features examining past connections to the same host or failed logins etc. The class label was transformed to the one-of-C class vector (41 original input variables plus 5 class variables where the first class denotes a non-attacking record). The following table (tbl. 3.2) shows the results of this experiment:
Table 3.2: Experimental Results: The experiments were performed on the training part of the DARPA intrusion detection dataset. The training sequences were run 100 times for every configuration to lower the randomness of the initial neuron distribution. The JGNG model configuration was tuned to have the same final count of neurons as the LVQ method. Error values are computed from cross testing where 90% of patterns are used for the training process and the other 10% is used for testing purposes. Mean error value donates how many test patterns have been incorrectly classified.

Another interesting feature of the GNG method is the possibility to visualize the structure of the network and information about the dataset distribution can be obtained through this structure. The method used for laying out the graph of the GNG network is discussed in the next chapter (see chapter 4.2) along with some other examples of GNG driven data visualization. Moreover, some other techniques of visualization has been used in this experiment. For example, class vectors are visualized as circles divided into colored arcs where every arc denotes one class variable and the radius of the arc is equal to the probability value.

Figure 3.5: Structure of the JGNG network trained to detect and classify intrusions. The network is projected into two-dimensions and colored arcs denote class part of each weight vector. The detail view shows one neuron which produces soft classification. If such neuron is the winner of the competition in the predicting stage, then the prediction will contain two most probable classes.
Chapter 4

Competitive network as supportive mechanism

4.1 Self organized preprocessing of feed-forward network

In the theoretical part of my thesis, I have introduced the feed-forward neural network model. Several methods how such network can be trained was also presented including gradient descent method and evolitional approach. Almost all algorithms used for training feed-forward networks start from random configuration. Weight vectors, thresholds and other attributes are usually initialized to random values although some constrains may be defined, such as interval or sign of weight vector elements.

However, there is a possibility to utilize the self-organization paradigm for initialization of the feed-forward network. This section is focused on how such initialization is done and what improvements are possible if the JGNG model is utilized within the process.

4.1.1 RBF networks

The Radial Basis Function Network or the RBF network [Buhmann03] is similar to the standard MLP network (in fact it is the feed-forward network) but a different transfer function is used for hidden units of the network. As the name of the model indicates, radial basis functions are used for transferring the potential into the output.

RBF networks usually have only two layers. The first layer is created by RBF units and the second layer (the output layer) contains standard linear or sigmoidal units. RBF units in the first layer are quite similar to standard units. The major difference lies in the transfer function and evaluation of the potential. While a standard sigmoidal neuron represents a hyperplane which divides its input space into two subspaces, an RBF unit covers only a local area of neuron input space; a hypersphere or a spherical potential field with the center in the weight vector position could be imagined. Outputs of the RBF layer are then processed by the second layer filled with standard neurons and the output vector of the network is generated. Formally, the firing mechanism of an RBF neuron may be described as follows [Sima96]:

\[ z = ||\xi - \mathbf{w}|| \]
\[ \varphi(z) = e^{-\lambda z^2} \]  

(4.1)

where \( \lambda \in \mathbb{R}^+ \) is the steepness factor of the RBF unit. The weight vector \( \mathbf{w} \in \mathbb{R}^n \) denotes the center of the RBF unit. There are several different ways for how the potential and the output can be evaluated and the threshold \( \theta \in \mathbb{R} \) can be also involved. As I found, it is possible to use the standard sigmoid function because the principle of the local behavior lies in the radial evaluation of the potential (notice that the sigmoid is flipped vertically, see eq. 2.6):
Figure 4.1: Firing mechanism of an RBF unit. In the standard Gaussian based method only the steepness is adjustable. The second method have two variables: the threshold (serves as the radius of a hypersphere) and the steepness.

\[ \varphi(z) = \frac{1}{1 + e^{\lambda(z-\theta)}} \]  

(eq. 4.2)

The following figure shows the difference in firing mechanism between these two approaches:

4.1.2 Adaptation of RBF network

The adaptation of RBF network is based on the same idea as the adaptation of any other MLP network. The goal is to set weights, thresholds etc. to values where the network will give correct answers. Obviously, the training set filled with training patterns is needed to reach such goal.

It is possible to employ the standard gradient descend method (backpropagation) and iterate through the training set repetitively and find the suitable configuration with the minimum error as usual. However, due to the nature of weight vectors inside RBF units, self-organization may be used within the training process of the network. Following example of simple RBF network will clarify that this is a correct assumption.

Motivational example

Let there be a simple RBF network of only one unit in each layer. Let the network has \( n \) inputs and one output (fig. 4.2). Let there be a training set of 2 patterns \( T = \{ (\xi, 0), (\xi, 1) \} \) where
\( \xi \in \mathbb{R}^n \) is an input signal and the second element is desired output for such input signal (see sec. 2.5).

The backpropagation algorithm will iterate through the training set and will evaluate a change \( \Delta w_i \) for every weight in the network. Let us focus on the first unit. The weight of the second unit is fixed to \( w_1^{\text{1}} = 1 \). The change of weights of the first unit is based on the standard gradient descend method (see eq. 2.8). When expanded, it goes as follows:

\[
\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \cdot \frac{\partial E}{\partial y} \cdot \frac{dy}{dz} \cdot \frac{\partial z}{\partial w_i}
\]

where

\[
\frac{\partial E}{\partial y} = (y^1 - d) \cdot \lambda^1 \cdot y(1 - y) \cdot w_i^1
\] (4.3)

\[
\frac{dy}{dz} = -\lambda \cdot y(1 - y)
\] (4.4)

\[
\frac{\partial z}{\partial w_i} = -\frac{\xi_i - w_i}{z}
\] (4.5)

The most interesting part is the last one (eq. 4.5). The whole weight change formula can be rewritten as:

\[
\Delta w_i = \eta (\xi_i - w_i) \cdot a, \quad a \in \mathbb{R}
\] (4.6)

With \( z, y, y^1, \lambda, \lambda^1 \in \mathbb{R}^+ \) (which is always fulfilled) we may follow this idea: if desired value \( d = 1 \) and output of the network \( y^1 < d \), then \( a \) is positive. Otherwise, if \( d = 0 \) and \( y^1 > d \) then \( a \) is negative. In different view, the algorithm is trying to move the center of the RBF unit towards the position of the pattern when the output of 1 is expected and vice versa. It manifests the very same behavior as the common competitive network (eq. 2.13), especially the LVQ algorithm in the second stage (eq. 3.2).

It is a known fact that the basic backpropagation algorithm might have some serious disadvantages (the local minima problem or the slow convergence problem). If we know that the adaptation of weight vectors is very similar to adaptation of the competitive network, we may use it and overcome these problems.

### 4.1.3 Three phase training

If the self-organization is involved, the training algorithm for RBF network is usually divided into three parts:

1. Adaptation of positions of RBF units in the first layer (self-organization of weight vectors).
2. Initialization of other attributes of RBF units (steepness and radius)
3. Adaptation of RBF steepnesses and radii and concurrent adaptation of neurons in the output layer (usually, the standard backpropagation is used)

The goal of the first stage of the algorithm is to place RBF neurons into the input space. The placement has to be driven by the distribution of training data because we expected that the the network will do the input/output transformation in clusters where input signals are situated. There are many competitive networks and methods which can be utilized for this step. It is better to use some advanced self-organizing network like the SOM algorithm because the
distribution of RBF units is crucial for latter stages of the training where no additional change of RBF units position is allowed.

The task of the second stage of the algorithm is to initialize the radius of each RBF unit. The radius of the unit is very important variable and affect the generalization abilities of the network after the training in finished. There may be problems with generalization, if areas defined by radii are small and otherwise, the local character of the RBF unit may be lost if radii are too high. There are several methods for how radii may be initialized. Some authors mention evaluation of the error function caused by inaccurate radii and minimization of such function [Sima96] but in practice, more simple approaches like evaluation of radius according to neighbors positions are often used. When the sigmoid transfer function is used (eq. 4.2), the threshold \( \theta \) represents the radius of the RBF unit. The radius of the unit \( u \) may be initialized as follows:

\[
\theta_u = \sum_{i \in \mathcal{U}(u)} \frac{\|w_i - w_u\|}{|\mathcal{U}(u)|}
\tag{4.7}
\]

The third stage of the algorithm is focused on adaptation of the output layer. Due to the simplicity of this task, the standard backpropagation may be used. Positions of RBF units are fixed in the third stage, and therefore, there is no need to backpropagate the error to the lower layer. However, it is necessary to adapt steepnesses and radii of RBF units. The standard backpropagation algorithm is suitable for such task.

4.1.4 Initialization of RBF layer with GNG and JGNG methods

As it has been described in the previous section, the quality of self-organization is crucial for the whole training process of the RBF network. The ongoing adaptation process is unable to correct poor initial distribution of RBF units. Many self-organization algorithms, unlike the RBF networks, learns from input signals only and no desired output is involved within the learning process. One may think of situations where such omission causes unsuitable initial distribution of RBF units. The following pictures shows two-dimensional example where such situation arose.

On the left side (fig. 4.3a), the standard GNG method has been used for the initial distribution of RBF units. Notice highlighted units that are positioned on the boundary between two distinct classes of training patterns. These units will be almost useless because the function realized by the network passes through a sharp change and it is not possible to fit any radial basis function with a meaningful radius in such position. On the right side (fig. 4.3b), the same dataset has been processed with JGNG method which utilizes class labels stored in training patterns. It is clearly visible that there are no units positioned on the boundary and all units may be used by the second stage of the RBF training. The following table (tbl. 4.1) contains results of the RBF trainining for both methods.

The results show that the JGNG initialization improved the training process of the RBF network. Also, the standard deviation values indicates that the JGNG method initialization is better in avoiding the randomness of the training process, i.e. fast convergence can be achieved even from an unfavorable starting configuration. The figures below (fig. 4.4) show final state of the RBF network when all stages of training are completed.
Figure 4.3: Difference between the standard unlabeled GNG distribution and JGNG distribution. Both methods were configured to run at maximum of 20000 iterations.

<table>
<thead>
<tr>
<th>RBF init method</th>
<th>Mean iterations</th>
<th>Std. dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNG run 1</td>
<td>1756.02</td>
<td>590.17</td>
</tr>
<tr>
<td>JGNG run 1</td>
<td>1136.58</td>
<td>178.10</td>
</tr>
<tr>
<td>GNG run 2</td>
<td>964.02</td>
<td>286.54</td>
</tr>
<tr>
<td>JGNG run 2</td>
<td>725.86</td>
<td>159.85</td>
</tr>
</tbody>
</table>

Table 4.1: Experimental Results: Experiments were performed on two-dimensional example dataset (fig. 4.4a). The training sequences were run 50 times for two different configurations (learning factor were tuned) to minimize the randomness of the initial attributes. The mean iteration column denotes the mean of iterations each method needs to converge to minimum cumulative error of 0.001.

4.1.5 Initialization of output layer of RBF network

In the previous section, a method for how the JGNG network can be used for improved initialization of the RBF layer has been presented. The improvement is based on utilization of class variables stored in every training pattern. One may also think of a possible initialization of the output layer in order to help the ongoing backpropagation algorithm converge more quickly. This section deals with possible solution.

The idea of initialization of the output layer units is quite simple. If the JGNG method is used for distribution of RBF units, every unit of the JGNG network can produce an output vector. An RBF neuron is constructed from every JGNG unit in the second stage of training. Output of the whole RBF network needs to be similar to the output of the JGNG network but more accurate. The JGNG network produces the output according to the Voronoi region of each unit. The RBF network will produce similar output but with soft boundaries due to the combination of neighboring units. If we know what output should be generated in the area around each JGNG unit then the only step which needs to be done is to prioritize weights that lead to RBF units where corresponding JGNG unit gives the same output as current output.
The following figure may clarify this approach:

Another attribute which has to be initialized is the threshold. If the threshold stays set to zero, the hyperplane realized by the neuron will pass through the origin of the input space which is useless. Therefore, the threshold needs to be set to suitable non-zero positive value, for example $\theta_u = \|w_u\|/2$. The pictures below show result of complete initialization process (RBF units weights and thresholds, output units weights and thresholds) of the RBF network run above the spiral dataset (fig. 4.6).

It is clearly visible that the initial state of the network follows the distribution of training patterns and their class values. The final backpropagation learning (fig. 4.6b) deals with minor inaccuracies and improve generalization abilities of the network, i.e. areas where no patterns are placed is giving “red” answer. Notice the distribution of patterns in the training set (compare to fig. 3.1b). Again, RBF units and their radii are highlighted as circles.

4.1.6 Experiments

There are two experiments presented in this section to confirm the possible improvement to the initialization process of the RBF network when the JGNG method is involved.

MNIST dataset

The MNIST handwritten digits database has been used for this experiment. The purpose of this experiment is to prove the fact that the JGNG driven initialization process of the RBF layer will provide better initial position for the fine backpropagation driven learning of the RBF network. The following table (tbl. 4.2) shows results of different initialization procedures:

The results show that there is a little difference between the standard GNG initialization and the initialization driven by the JGNG method. Such result can be clarified by the high dimension of the dataset space where the distribution of training patterns (i.e. images of handwritten digits) is very sparse. In such situation the influence of class label is not very strong (see sec. 3.2.2).
Figure 4.5: Weight prioritisation in the output layer of the RBF network. Weights are set according to class variables stored in JGNG units. Three example inputs ($\xi_1, \xi_2, \xi_3$) are shown along with their projections to the input space where output units operate.

Table 4.2: Experimental Results: Experiments were performed on the MNIST dataset [MNIST98]. The training sequences were run 50 times to minimize the randomness of initial attributes. Three different initialization methods were used. The out layer init column denotes if the output layer of the RBF network has been initialized according to the class attributes stored in JGNG units. The mean error column denotes the mean of distortion error each method achieved within 30 iterations through the training set.

<table>
<thead>
<tr>
<th>Init method</th>
<th>out layer init</th>
<th>Mean error</th>
<th>Std. dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNG</td>
<td>N/A</td>
<td>0.251</td>
<td>0.015</td>
</tr>
<tr>
<td>JGNG</td>
<td>No</td>
<td>0.247</td>
<td>0.016</td>
</tr>
<tr>
<td>JGNG</td>
<td>Yes</td>
<td>0.192</td>
<td>0.004</td>
</tr>
</tbody>
</table>

and the GNG initialization is sufficient enough. On the other hand, the initialization of the output layer of the RBF network which is created by standard sigmoidal neurons sped up the backpropagation stage. The standard deviation value proves the fact that the system starts in very similar state and there are fewer randomly good or bad initial configurations (third row in tbl 4.2). Such improvement can have practical impact to experiments done with this initialization method where the researcher need not to do experiments repetitively to achieve “a lucky case” of better initial configuration.

**Intrusion detection dataset**

The next experiment has been run above the Intrusion Detection dataset. The dataset has been previously used for experiments with the JGNG network (see sec. 3.3.3). The following table shows the results of the RBF training with the GNG/JGNG initialization process. The purpose of this experiment is the same as in the previous experiment: to measure the impact of the JGNG driven initialization of the RBF network.
Figure 4.6: JGNG initialization of RBF network. On the left side, there is an example initialization which was set before fine teaching driven by backpropagation was started. The final post-backpropagation state of the same network is shown on the right side.

![RBF initial state](image1) ![RBF final state](image2)

Table 4.3: Experimental Results: Experiments were performed on the Intrusion Detection dataset [DARPA98]. The training sequences were run 50 times to minimize the randomness of initial attributes. Three different initialization methods were used. The out layer init column denotes if the output layer of the RBF network has been initialized according to the class attributed stored in JGNG units. The mean error column denotes the mean of error each method achieved within 30 iterations through the training set.

<table>
<thead>
<tr>
<th>Init method</th>
<th>out layer init</th>
<th>Mean error</th>
<th>Std. dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNG</td>
<td>N/A</td>
<td>0.105</td>
<td>0.010</td>
</tr>
<tr>
<td>JGNG</td>
<td>No</td>
<td>0.054</td>
<td>0.006</td>
</tr>
<tr>
<td>JGNG</td>
<td>Yes</td>
<td>0.047</td>
<td>0.005</td>
</tr>
</tbody>
</table>

The results (tbl. 4.3) show noticeable difference between the GNG and JGNG driven initialization method in the meaning of speed of the ongoing backpropagation learning. If we take into account that both methods prepare the same number of RBF units and the JGNG method sped up the backpropagation learning process then this may indicate that the dataset space is populated densely and the standard GNG learning probably put RBF units into areas where the units have little use (similar to fig. 4.3).

On the other hand, there is almost no difference when the additional output layer initialization is applied. Further analysis showed that the requested mapping from RBF units to desired output had minor importance and the most of the learning effort has been used to adapt RBF units.
4.2 Visualization of high-dimensional data using GNG

The following section focuses on the problem of the visualization of high-dimensional datasets, common methods used for this task and the possibilities of using the GNG algorithm as a supportive mechanism for the visualization of high-dimensional data.

In the previous chapter, some visualizations of the GNG network were presented along with datasets above which the network was trained. However, most of the examples presented contain only low-dimensional data (2D or 3D), and therefore, there is no problem with dataset visualization. Only one example (fig. 3.1) shows a high-dimensional dataset projected onto a two-dimensional surface. Later in this section, the technique used for such visualization is addressed and some other examples of the visualization of high-dimensional datasets are given.

4.2.1 Importance of visual information

Visualizing of items and processes around us is a natural human behavior and it is a part of almost any human activity. Sight is a fundamental human sense and the brain is adapted for receiving and interpreting graphical information. Another important power of the human brain is the ability to effectively interpret graphical analogies. In such way we are capable of depicting completely abstract objects such as vectors or algorithms. The graphical interpretation of these objects is usually very sophisticated and even an untrained person is capable of understanding basic principles very fast, if a graphical aid is provided. Such facts are well known and obvious but it is appropriate to mention them because even in the present some people underestimate the power of visualizing their results. On the other hand, there are cases where the graphical form dominates provided information, and therefore, it is vital to focus on the content and choose the form according to it.

In my thesis, I'm trying to show that graphical information is very powerful and I use a scheme or a figure whenever I feel that the picture will clarify concepts and ideas I would like to present. Sometimes, high-end 3D display or even a interactive panel embedded into the page will be very helpful and in such places I rely on the imagination of the reader.

Unfortunately, all visualization techniques still end up on a surface. There are devices capable of 3D visualization, but practically, we are still bound to 2D images. Hardware is not the only thing constraining us; we are bound by the principle of our vision. The human eye is basically a two-dimensional sensor and many things like reading and writing are derived from this fact. Obviously, we are capable of sensing the third dimension through indirect information (two-eyed vision or perspective distortion) but our two-dimensional imagination is much better than our three-dimensional one. Fortunately, we are very good at combining various kinds of information together, and therefore, there are many methods for how high-dimensional data could be processed and projected onto a two-dimensional surface with moderate information lost.

4.2.2 High-dimensional visualization methods

There are many different ways in which high-dimensional data can be visualized [Grinstein01]. The aim of this section is to introduce some of them and to give the reader basic orientation in this topic.

The most simple method used for visualization of 2D or 3D data is probably the scatter plot visualization. A scatter plot is a point projection of the data into a classic Cartesian system.
Of course, if 3D data is plotted, some projection technique such as perspective projection or axonometric projection has to be used. The scatter plot visualization is very natural and it is possible to enrich it in various ways (color or shape of points, interactive rotation, etc.).

Matrix of scatter plots

There is a simple enhancement of scatter plot visualization when high-dimensional data (i.e. with a dimension of 3 or greater) is plotted. It is a matrix of scatter plots which displays all possible pairwise combinations of dimensions or coordinates with a single scatter plot. For \( n \)-dimensional data, this yields \( n(n-1)/2 \) scatter plots with shared scales, although usually \( n^2 \) scatter plots are displayed (a histogram may be plotted when a pair of identical dimensions is processed). All scatter plots are distributed a in typical matrix fashion, and consequently, one can visually link features of one scatter plot with features on another, which greatly increases its power. Additionally, the scatter plots can be positioned in a non-array format (circular, hexagonal, etc.). An example iris plants dataset is shown below as a matrix of scatter plots (see sec. 4.3.1 for dataset details).

```
sepal width

sepal length

petal width

petal length

sepal length

sepal width
```

![Figure 4.7: Matrix of scatter plots – four-dimensional UCI Iris Plants Database](image)

It is clear that the matrix of scatter plots is useful only with a dataset of moderate dimension because this technique is quite space-consuming. However, almost no information is lost within the process of visualization.

Parallel coordinates

Another visualization method uses parallel axes instead of perpendicular axes to represent dimensions of a multidimensional dataset. A polyline containing \( n-1 \) lines positioned at the appropriate dimensional values connects the axes to represent an \( n \)-dimensional point. The next figure (fig. 4.8 left) shows this method when the same the iris plants database is used.
Also, sometimes it is better to arrange axes in circle where every $n$-dimensional point forms a polygon intersecting each axis in an appropriate position (fig. 4.8 right).

Figure 4.8: Parallel and spherical coordinates – four-dimensional UCI iris plants Database

**SOM, PCA, RadViz**

Until now, all presented methods display $n$-dimensional points directly, at most, standardization of values is applied. However, there are methods that perform some kind of dataset analysis before visualization. Such analysis can have very different forms and results. For example, Principal Component Analysis (PCA) \cite{Hotelling33} tries to find so-called principal components through a series of orthogonal transformations of the original dataset. These components are sorted along with decreasing variance in the direction of the component and the number of principal components is lower than the dimensionality of the dataset. Usually, a scatter plot is used for displaying the results of the PCA.

Figure 4.9: Self-organizing map – iris dataset. Data instances are projected onto reference vectors. Reference vectors are distributed as a rectangular grid. The spatial neighborhood between the original data is reflected through the proximity within the grid. The original 4-dimensional positions of reference vectors are described by arcs around each unit where every arc denotes the proportional value of the appropriate attribute. The color of each unit is evaluated as the most mapped class variable borrowed from the LVQ method (see. eq. 3.1).
The SOM algorithm may be understood as another analytical method of preprocessing before the visualization of high-dimensional data. If we look at the SOM algorithm from the visualization view, then it performs dimensionality reduction into two or three-dimensional space which can be displayed on a surface. Every n-dimensional point is projected to a reference vector (the weight vector of neuron) and the grid of neurons is visualized in a common way. The picture above (fig. [4.9]) shows the SOM driven visualization of the iris dataset.

There are several possible visualizations of SOM units and relations between neighbors. For example, the $U$-matrix visualization involves Euclidean distances between direct neighbors and these distances are visualized as grayscale map [Ultsch90]. Moreover, it is possible to utilize the $U$-matrix for clustering purposes by dividing the SOM grid over connections that exceed specified length.

Another interesting solution of high-dimensional visualization is the RadViz [Hoffman99] method. This method performs very simple but effective preprocessing where all $n$-dimensional points are attached to dimensional anchors distributed around the perimeter of a circle. Attachments are flexible and their stiffness is set according to values in the original dataset. Each point is displayed in a position where the sum of forces generated by connections are balanced. In such way, the original position of the $n$-dimensional point is lost but the distribution of the given dataset is clearly visible.

4.2.3 Growing Neural Gas

The visualization method of an $n$-dimensional dataset driven by the Growing Neural Gas algorithm has similar attributes to the methods described in the previous section. In fact, it is a competitive method which performs a very similar relation-based analysis to the SOM algorithm but it does not form any fixed structure, such as a grid. On the other hand, it generates a graph which preserves spatial relationships between data points.

However, we must not forget that the idea of visualizing the dataset through the structure of the GNG network completely omits the original positions of data points. One may think if there will be any useful information left if such omission is allowed. To answer this, we have to think about possible structures that the GNG network may generate. For example, an interesting feature of the GNG graph is the number of isolated sub-graphs that may denote clusters in a dataset. The shape of sub-graphs is another useful feature. Also, some other attributes may be utilized when the graph is visualized, such as the number of mapped inputs that are projected onto each graph node (the GNG neuron). After all, there are many interesting features even when positions of neurons are not involved. Experiments at the end of this section will show some of them.

Currently, our problem is reduced to a graph plotting problem. The last question is how positions of graph nodes should be determined if a "good arrangement" of a graph is sought. Fortunately, force-based or force-directed techniques can handle such a situation with more than satisfactory results.

4.2.4 Particle/spring graph model

The Force-Based Algorithm (FBA) is usually cited as a graph drawing algorithm [Brandes01]. The main idea of the FBA is a construction of a pseudo-physical model of a graph. If such model is simulated from some random initial state, it converges quickly to a balanced state i.e.
a state with minimal energy. The graph is drawn when the balanced state is reached.

The physical model of the graph is very simple (see sec. 5.2 for detailed definition). Graph nodes are modeled as electrically charged particles and edges between them are flexible springs. Particles are shapeless, and therefore, the motion is linear and no rotational part of the motion is needed. At every simulation step, the resulting force is determined for every particle and the particle is moved according to the Newton’s laws of motion. Formally, forces and simulation are driven by the following equations.

\[ F = F_r + F_a \] (4.8)

The attractive force affecting particle \( p \) at position \( x_p \) is a sum of attractive forces generated by connections to its neighbors:

\[ F_a = \sum_{i \in U(p)} \alpha(x_i - x_p) \] (4.9)

The repulsive force affecting particle \( p \) is derived from Coulomb’s force where all particles are charged with the same charge i.e. they repulse any other:

\[ F_r = \sum_{i \in P} -\beta \frac{\epsilon_p \epsilon_i}{\|d_i\|^2} \hat{d}_i \text{ where } d_i = x_i - x_p \] (4.10)

The simulation of the physical model is driven by the simple Euler integration method:

\[ \Delta v = a(t) \Delta t, \quad \Delta x = v(t) \Delta t \]
\[ x(t + 1) = x(t) + \Delta x, \quad v(t + 1) = v(t) + \Delta v \] (4.11)

4.2.5 Attributes of force-based layout of GNG graph

The method described above gives us the ability to visualize a dataset of arbitrary dimension. A kind of dimensionality reduction is performed because the graph is projected into two-dimensional or three-dimensional space. Still, one question remains. How well does the reduction process preserve the structure of the original data?

The visualization process is based on a graph generated by the GNG algorithm. The graph is compounded by neurons (\( n \)-dimensional reference vectors) and edges. If two neurons lie near each other and there is no other neuron in close proximity, there will be a connection between them (see sec. 2.6.3). When the graph is extracted from the original \( n \)-dimensional weight space into the target 3D space or 2D surface, all positions of neurons are lost. However, spatial relationships are preserved in connections. Now, we are looking for a graph plotting algorithm which aims for short edges between vertices. The condition of short edges means that connected vertices (neurons) will lie close to each other and the structure of the visualized dataset will be preserved. The force-based algorithm meets such condition by the attractive force definition (eq. 4.9) which tries to minimize the distance between connected vertices.

Another spatially related attribute which may be taken into account is the density of the dataset. The GNG algorithm is optimized to build dense neuron clusters where the dataset is dense and vice versa. When the positions of neurons is omitted, the density information is lost. Nevertheless, if there is a sparse cluster in the original weight space, edges emanating from neurons are generally longer. These lengths can be stored and used when the force-based layout
is performed. The best way to use the lengths is to establish an indirect dependency between the original edge length and the spring factor $\alpha$. A longer edge in the original weight space will cause a less stiff spring in the force-based layout thus the vertices will be less attracted and the distance between vertices will increase.

Unfortunately, there are disadvantages. Probably, the most tough one is the problem of edge crossing when a typical two-dimensional display device is used. Heavily connected graph may not result in well-arranged graph by any mean. Even worse, if the force-based algorithm is simulated in two dimensions the local minimum problem will arise for sure [Brandes01]. The simulation will probably converge to undesired local equilibrium where needed spatial relations are not satisfying.

However, the problem is partially solvable by moving into three dimensions. The system will gain another degree of freedom which results in better convergence and still, it will remain human readable when appropriate method of visualization is used (such as perspective projection). An interactive visualization tool with a kind of orbital camera may be a great help in such situation.

Another possible solution to the problem of the local equilibrium is the concurrent execution of the GNG algorithm and the relaxing process of the force-based layout. The GNG algorithm starts with only two units connected together. Such situation is easily solvable. When the GNG algorithm advances and the generated structure grows into dense graph, the concurrently running layout process results in better arrangement. The local equilibrium problem is suppressed by growing graph where any new unit causes stress to the relaxing process and it may pull out the relaxation from undesired local minimum.

### 4.3 Examples of GNG driven high-dimensional visualization

To test the basic behavior of described method we can use the dataset presented in the first chapter (fig. 2.12). The following figure (fig. 4.10) shows the GNG structure after 40000 iterations of the learning process. The structure is visualized using force based graph plot and it is clearly visible that the structure of the dataset preserved the omission of units positions and the shape of the dataset (a twisted H letter) is recognizable.

![Figure 4.10: Force-based layout, the twisted-H three-dimensional example. No position information was used during the visualization process. The spatial relation is carried within edges; the structure of the dataset preserves.](image)

---

1Obviously, this example is very basic because there is no dimensionality reduction and it is included more or less for presentation purposes.
4.3.1 Iris plants database

The next example is very well known UCI Iris Plants Database [Frank10]. The dataset contains three classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two; the latter are not linearly separable from each other. Every instance has four continuous attributes – the input space is four-dimensional. The dataset has been already presented as example dataset for other methods of high-dimensional visualization (fig. 4.7, fig. 4.8); one may compare the quality of presented visualization. The following pictures show the GNG driven visualization of the dataset plotted into three dimensional space and viewed from two different angles (fig. 4.11).

![GNG driven visualization of the UCI Iris Plants Database](image)

Figure 4.11: GNG driven visualization of the UCI Iris Plants Database. Data instances is visualized through neurons of the GNG network. The color of every GNG unit is derived from the most mapped class retrieved during the learning process.

The main difference between the GNG driven visualization and other methods presented above is that the data is plotted through reference vectors of the GNG neurons i.e. no direct data instances is visualized. Hence, it is not possible to show data labels (points colors) directly and a different approach has to chosen. A simple but useful method used in the LVQ algorithm was found suitable for this problem (see eq. 3.1). In different words, each unit’s color is derived from the most mapped class value evaluated during the GNG learning process.

A disadvantage of GNG driven visualization is that the exact distribution of variables is traded of for the ability to visualize dataset of almost any dimensionality. When the GNG driven visualization is used, the distribution of dataset is described by the spatial neighborhood of data instances. When a dataset has a low-range level of dimensionality (e.g. the iris dataset) it may be more useful to use some other method of visualization, such as parallel coordinates. But, if the dimensionality of the dataset is high (a few hundreds for example) it is not possible to use almost any method which plots attributes directly. The next example tries to show how the visualization such dataset may look.
4.3.2 MNIST dataset

The dataset used for the next example is the same dataset which has been used in the previous chapter (see section §3.3.2). This example aims for the visualization of a dataset with very high dimensionality (a square raster of 28 pixels per row i.e. 784 pixels total). If one think of possible methods of visualization of such dataset, then there is always picture-driven approach because instances are actually gray scale pictures. On the other side, there are thousands of pictures in the database thus some advanced way of organization has to be used. One can group pictures according to their labels (i.e. actual numbers appearing on the picture). Also one may think of grouping pictures which looks similar. All these attributes can be achieved by the GNG driven visualization which is shown on the following picture (4.12).

Figure 4.12: GNG driven visualization, the MNIST dataset example. The position of every reference vector (i.e. GNG neuron) is depicted. Two groups of spatially related neurons are highlighted where each neuron represents different numbers that look similar.
Unfortunately, in this example, the generated structure of the GNG network is very complicated and the distribution is not as clear as in the previous case. At this point, an interactive visualization tool with ability to rotate/zoom/pan view and to examine the structure from different angles becomes crucial. Eventually, such visualization tool can be useful for validation of the trained JGNG network when the classification task is requested. The problem of understanding the inner structure of any artificial network is well-known and it is always a positive fact if there exists a user-friendly visualization method for particular neural network structure which can point out actual configuration after the training process is done. Moreover, this approach has been successfully implemented into the ANN Tool developed by myself (see appendix section 8.1) and it has been tested during various experiments I have run so far.

4.3.3 Intrusion detection dataset

The next visualization example shows the intrusion detection dataset (see sec. 3.3.3). The dataset has been previously used for testing the JGNG classifier performance and also, the force-based visualization technique has been already used to present results of the classifier (see fig. 3.5). However, let us imagine the situation where data is not labeled by proper intrusion type and only input attributes are known (in other words, it is the standard initial position for unsupervised learning). The following figure shows the GNG driven visualization of the unlabeled intrusion detection dataset:

Figure 4.13: GNG driven visualization, intrusion detection dataset example. The structure of the dataset can be examined through the structure of the GNG network. Every weight vector is visualized as a raster of $6 \times 8$ pixels.

It is visible that the dataset contains several clusters represented by branches emanating from the core structure. Moreover, these branches describe interesting parts of the dataset. For example, if one compare the highlighted cluster with actual labels, then 80% of no-intrusion records are mapped to neurons within this cluster.

\footnote{Obviously, the interactive visualization tool can show this distribution better.}
Chapter 5

Physically based self organization

In the previous chapter, a visualization method for high-dimensional datasets was introduced. The method combines the physically based simulation of the particle/spring system and the competitive network.

In this chapter, the usage of the particle/spring system continues, and it is consequently described in greater detail. A new approach to competitive learning is proposed. The standard competitive network is altered and turned into the physically based particle system driven by motion equations. First, a pseudo-physical model of a simple competitive network is described. Differences and similarities between the physically based model and the standard approach are examined and several favorable attributes of the physically based approach are highlighted. Next, the physically based model of the well known Self-organizing Map is constructed. Again, some promising attributes of the new approach are pointed out, including destructible connections between units and ongoing benefits of such behavior. At the end of the chapter, experiments are reported. The method of measuring the quality of the trained competitive network is suggested and the comparison between the standard approach and the physically based method is reported through several examples.

5.1 Physically based approach

There are vast cases where a kind of physically based system is used or/and simulated. Obviously, there is a need to simulate many real physical systems; for example, body dynamics within machinery production or in the medicine. Sometimes, these systems can be very large (the atmosphere in the weather simulation) and there are many directives for how the real system should be simulated and what conditions need to be met (e.g. integration solvers, discretization rules etc.). On the other hand, it has been shown that sometimes it is very useful to use real world physical systems as an inspiration for solving very abstract problems. For example, the global optimization problem has several physically based solutions, such as the simulated annealing \textsuperscript{Kirkpatrick83} \textsuperscript{Cerny85} or the particle swarm optimization \textsuperscript{Kennedy95}. Another good example is the image segmentation problem where several physically based methods have been published, for example, the flooding technique \textsuperscript{Roerdink01}. Nevertheless, there are neural networks inspired by physical processes. The next section is dedicated to one of them.

5.1.1 Simulated annealing & Boltzmann Machines

Simulated annealing is a probabilistic solver for the global optimization problem, i.e. the problem of finding the global optimum of given function. The inspiration comes from the metallurgy area where annealing means the technique of repetitive heating and controlled cooling of the material to improve desired attributes (the size of crystals and number of defects). The heating of the material causes the increase of internal energy and the atomic structure become unstable. When
the material is cooled, it is possible that it finds the configuration with lower internal energy (i.e. with lower internal tension) which results in better physical attributes.

The SA algorithm is quite similar to this physical process. In every step, the current solution is replaced by a random neighboring solution with a given probability. The probability of such move depends on a global parameter $T$ – the temperature of the system which decreases over time, and a difference between the current solution and a nearby solution. There are some rules about how the temperature $T$ and the energy difference ($e = E(t)$; $e' = E(t+1)$) are involved. Usually, the probability of moving the solution is given by a function $P(e, e', T)$. Basically, if $T$ is large, then the $P \approx 1$; this denotes random behavior at high temperatures. Also, if $e' > e$ then $P > 0$ i.e. it is possible to move “uphill” to worse solution. Such condition prevents the algorithm being stuck in local minima. However, the probability of uphill move has to be lesser then the probability of moving to better solutions, especially at low temperatures. Another significant attribute of the SA algorithm is the cooling schedule. Various progressions may be used including repetitive heating and cooling and usually, the schedule is tied to the current application of the algorithm. In such way, an acceptably good solution can be found in a fixed amount of time i.e. until the temperature reached zero.

When the SA algorithm is applied to the training process of the Hopfield Network [Hopfield82] a probabilistic Hopfield Network can be constructed. Such model has been published as the Boltzmann Machine [Ackley85]. The major advantage of the probabilistic SA driven training process is ability to unstuck the network from undesired local minimum due to the probabilistic principle of the state change. The training process of the Boltzmann Machine includes two phases which are switched iteratively. In the “positive” phase the network inputs are fixed to chosen training pattern values. In the the second ”negative” phase, the network is allowed to run freely. The weight change is determined according the stable states reached at the end of each phase. Unfortunately, the Boltzmann Machine is rather theoretical model and there are serious problems when it is applied in practice. Due to the possible inter-connections the model is very computationally expensive and practically unusable. However, a restricted version the the model exists – the Restricted Boltzmann Machine [Hinton02] which can be applied on real problems.

One of the most interesting property of the Boltzmann Machine (in connection to my re-search) is that the simulation of the the system converge to the thermal equilibrium where the states of network units is fixed. The equilibrium principle is used widely in the area of physically inspired models including majority of particle/spring systems and also further presented physically based competitive network uses this principle.

5.2 Particle/spring system

Particle/spring systems are created with lumped masses, called particles, which are connected by elastic springs. There are various forces affecting the system, such as attractive forces generated by springs, local forces generated be particles or external forces from static or dynamic sources. Usually, the sub goal of the method which utilizes the particle/spring system is to find equilibrium state where all forces are balances, i.e. the system is in the state with the minimal kinetic energy. The equilibrium state can be found by simulating the system over time from defined initial state. Generally, there are two different solution solvers of such task; explicit solver or implicit solver. Explicit solver tends to find the solution from the current state of the
system where the implicit solver rather solves the equation involving the current and the latter state. Both methods can be used but the explicit method often fails when the system becomes too “stiff”. The usage of both methods is discussed in the following text but first, the particle system has to be defined:

A particle/spring system $PS$ can be defined as a triple:

$$PS = (\mathcal{P}, \mathcal{S}, \mathcal{F})$$

(5.1)

where $\mathcal{P}$ is a set of particles, $\mathcal{S}$ is a set of connections (springs) and $\mathcal{F}$ is a set of external forces. Every particle $p$ is defined by its position $x$, velocity $v$ and mass $m$:

$$\mathcal{P} = \{p_1, \ldots, p_n\} ; \quad p = (x, v, m)$$

(5.2)

Every spring within the system realizes a connection between two particles and it is assigned a stiffness constant $k \in \mathbb{R}^+$. Is it assumed that connections in the system are undirected hence the relation which realizes them is reflexive (see sec. 2.6.3 for similar definition):

$$\mathcal{S} = \{s_1, \ldots, s_m\} ; \quad s = (c, k) ; \quad c \in \mathcal{P} \times \mathcal{P}$$

(5.3)

The set $\mathcal{F}$ contains forces affecting particles. Every particle can be affected by some external global force $F_g$, force generated by spring connections $F_s$ and some optional particle dependent force $F_p$ (e.g. inner-particle repulsion):

$$\mathcal{F} = \{F_1(p_1), \ldots, F_n(p_n)\} ; \quad F(p) = F_g + F_s(p) + F_p(p)$$

$$F_s(p) = \sum_{i \in \mathcal{U}(p)} -k(x_p - x_i) ; \quad i \in \mathcal{U}(p) \Leftrightarrow ((p, i), k) \in \mathcal{S}$$

(5.4)

Such system is simulated in time using well-known kinematics and mechanics laws of motion. Since particles are lumped masses with no shape, motion is linear only; no rotation part of motion is needed:

$$\frac{dx}{dt} = v ; \quad \frac{dv}{dt} = a ; \quad F = ma$$

(5.5)

Generally, to predict (or evaluate) positions of all particles in arbitrary time one has to solve a special case of $n$-body problem which still remains analytically unsolvable [Diacu96]. However, a numerical solution of such system is used widely and various methods can be used for this purpose. The most basic method is probably the Euler Forward Method which is quite straightforward and easy to implement:

$$x(t + 1) = x(t) + v(t)\Delta t$$

$$v(t + 1) = v(t) + a(t)\Delta t$$

(5.6)

Unfortunately, the method is not accurate and it is unstable. Serious problems will appear if the simulated system is “stiff” and the method need to use very small $\Delta t$ to avoid divergence (the simulation literally explodes). There are other, more sophisticated, numerical methods usually used within the problem domain, such as the explicit Runge-Kutta fourth order (RK4) method [Ascher98], but in certain cases, if the task does not demand high accuracy, the Euler method may be used as well.
5.2.1 Applications of particle/spring system

There are many applications where the particle/spring system can be involved. Some of them are connected to real physical systems in order to approximate complicated structure of real objects. For example, particle/spring systems are often used for simulation of soft-body dynamics or cloths [Kaldor08, Selle08]. Another popular application is the skeletal simulation a.k.a the rag doll simulation. The system can be extended in various ways, e.g. for the rag roll simulation, torsion springs may be added to achieve better behavior of the model.

On the other hand, there are applications of particle/spring systems that are not directly connected to real world physical systems and where the behavior of particle/spring system was found useful. In the previous chapter, the RadViz algorithm which use physically based springs has been introduced and very popular graph plotting algorithm driven by particle/spring system has been also reported. Another indirect application of particle/spring system is for example the Particle Swarm Optimization [Kennedy95] method where physically based model deals with a quite abstract problem of global optimization.

Study of these topics motivated me for the research of possible incorporation of particle/spring system into the area of competitive learning. The next section is focused on the definition of physically based system that will manifest the very same behavior as the competitive network.

5.3 Model of physically based competitive network

The standard learning process is based on an adaptation of the neuron’s weight vector according to the input signal passed to the network. An input signal is passed to the network in every learning step, the winner is found and the adaptation (i.e. the change of the weight vector) is evaluated. In the physically based approach we want to achieve the same effect through forces affecting the particle system (eq. 5.4).

My model is a simple particle/spring system. I assume that the neuron weight vector is represented by the position of the particle. It is also assumed that every input signal from the training set may be a source of a distance dependent force. The sum of forces from every input signal to every particle (neuron) may have a similar effect as the standard adaptation rule (eq. 2.13). The physically based learning process stands for a simulation of the particle system over time (eq. 5.6). In every time step, the sum of all forces affecting the particle is evaluated and the change of its position becomes an adaptation of the neuron.

There is a significant difference between the standard learning process (sec. 2.6.2) and the physically based simulation described above. The standard learning process is incremental in the meaning that there is only one input signal processed in every time step. On the other hand, physically based learning evaluates the influence of all input signals within a single time step. Nevertheless, there is a similar approach used widely in neural computing called the batch learning where the adaptation is evaluated after all input signals are processed (sec. 2.5.2). My physically based learning process is the alternative of such batch learning method.

5.3.1 Winning neuron problem

The common principle of competitive networks is the competition of neurons. The competition is usually based on some kind of metric, typically the Euclidean metric (eq. 2.12). Unfortunately,
there is no simple physical model which has exactly the same behavior as the competition of neurons and individual adaptation of the winner. Nevertheless, there is a way to mimic such behavior by introducing additional forces to the system. The first is an attractive force between the input signal and the particle (neuron). This attractive force has to be distance dependent and it should affect the winning neuron (the closest one) the most, thus it needs to be inverse to the distance between the neuron and the input signal.

The attractive force is not sufficient to recreate the winning neuron behavior of the standard model. The standard model is so-called order dependent where the winning neuron takes a whole adaptation or most of it; the second neuron takes less of the adaptation etc. (see sec. 2.6.3). The attractive force will affect only close particles but two particles within the same distance will be attracted with the same force. We need to penalize the second nearest neuron somehow. The inter-particle repulsive force is the influence we need. Such force will cancel the attractive force (the adaptation) if there is another particle (neuron) in a direction of the input signal.

There are some other differences between the standard model and the physically based model such as the effect of particle velocity. The explanation of these differences along with a more detailed view of the physically based adaptation is given below.

### 5.3.2 Forces definition

At it was explained above, the model is based on simple particle system. The biggest difference is probably the fact that the common particle system is simulated in two or three dimensions. In case of our model, if we assume that the particle position is equal to the reference vector of the competitive neuron, therefore the dimensionality of the system could reach any positive natural number. Luckily, there is no need to change the principle of the particle system; it is applicable to arbitrary dimension.

Formally, the force that affect any particle is a sum of three elements. Every particle $p$ in the system is attracted to all input signals at once, i.e. every input signal $\xi_i$ stands for a source of distance a dependent force:

$$F_a(p) = \sum_i \frac{g}{\|d_i\|^h} \hat{d}_i ; \quad d_i = \xi_i - x_p ; \quad g, h \in \mathbb{R}^+ \quad (5.7)$$

The introduction of such force brings us a minor problem of creating a singularity in every input signal position. There is a simple solution of this problem – introduction of an additional short-range repulsive force between the input signal and the particle. Without such modification it is likely to happened that the neuron approaching towards the input signal position will be ejected out with very high velocity.

$$F_a(p) = \sum_i \left[ \frac{g_1}{\|d_i\|^{h_1}} - \frac{g_2}{\|d_i\|^{h_2}} \right] \hat{d}_i ; \quad g_1, g_2, h_1, h_2 \in \mathbb{R}^+ \quad (5.8)$$

As it was mentioned before, the inter-particle repulsive force has to be added to the system. The definition of this force is similar to the attractive force (eq. 5.7) but it is generated between particles. The inter-particle repulsive force affecting particle $p$ is defined as:

$$F_r(p) = \sum_{i \in \mathcal{P}, i \neq p} \frac{r}{\|d_i\|^2} \hat{d}_i ; \quad d_i = x_i - x_p ; \quad r \in \mathbb{R}^+ \quad (5.9)$$
The figure below (fig. 5.1) shows the progression of $F_a$ according to the particle/neuron distance where $g_1 > g_2$ and $h_1 < h_2$. The inter-particle repulsive force $F_r$ is also plotted.

Figure 5.1: Progression of the attractive force $F_a$ in dependence on a distance. Particles are attracted to the input signal ($\alpha$) but within the short range, the repulsive force ($\beta$) will prevail.

5.4 Physically based learning

The physically based learning is the simulation of the particle system over time. At the beginning, initial positions of all particles in the system are random within the meaningful interval e.g. within the range of the training set. Initial velocities are set to zero vectors. The standard learning process advances in discrete time steps of fixed length and the same approach is used in the physically based approach. The simple Euler method (eq. 5.6) was found satisfactory for our purposes. As the simulation advances, the particle system tends to converge to the configuration of minimal kinetic energy, i.e. to the configuration where forces affecting particles are balanced. We believed that such configuration will be also the configuration where the distortion error of the competitive network is minimal or small enough.

5.4.1 Equilibrium state

Generally, there are two major stopping criteria for the competitive learning. The first is simply stop the algorithm after a fixed number of steps regardless to the resulting structure of the networks. Such condition look quite primitive but practically, it is sufficient in many cases. The second approach is usually based on some measure, for example, the distortion error value. If the error gets lower then some specified value, the algorithm is stopped. Eventually, one may observe the progress of the distortion error and stop the algorithm if the error decrement is not sufficient any longer. Fortunately, the distortion error value remains meaningful when the standard competitive learning is replaces with the physically based approach.

If we examine the state of the system when the the distortion error stops decreasing it is clear that in such state the positions of networks units are fixed. In the standard competitive model, it rather means that units oscillates around some position. The size of this oscillation is tied to the number of units of the network and it also depends on the quality of particular learning case (see fig. 2.9 left – $\alpha$ cluster). We may call such state as the equilibrium state.

The situation is similar when the particle/spring system is used. The definition of the equilibrium state remains the same; particles are fixed in their positions. However, there is no

\[1\text{Actually, there is a common definition of the equilibrium state from classical mechanics: A system of particles is in static equilibrium when all the particles of the system are at rest and the total force on each particle is}

60
guarantee that such equilibrium state exists. For example, imagine a system containing only two unconnected particles repulsing each other with no additional force involved; the particles will move apart infinitely. The following text deals with this problem by analysis of the global force field generated by various configurations of inputs signals and positions of particles and it is shown that the equilibrium state is reachable in almost every case. Furthermore, to ensure the existence of the equilibrium state, some attributes of the particle/spring system have to be tuned.

One input signal configuration

Imagine the situation where the training set contains only one input signal. The force field generated by this signal ensures the existence of equilibrium state by definition. If we analyze the progress of the attractive force (fig. 5.1 – solid line) and if we take into account the indirect distance dependency of this force, then it is clear that there is an equilibrium point in the position of the input signal. To be more accurate with respect to forces definitions (eq. 5.8), it is better to say that there are an infinite number of equilibrium points around the input signal’s position (fig. 5.2a – a black circle).

![Figure 5.2: Force field visualization: The physically based learning process generates the force field throughout the weight space of the network. The force field is visualized as usual and additional information of force size in particular point is is plotted as gray scale value. Also, a possible trajectory of one particle is shown to explain unstable equilibrium point.](image)

Training set with finite number of input signals

The existence of equilibrium points is ensured by the definition of attractive force even if a set of input signals is involved. If one take into account a sum of input signals, then these signals can act with or against each other. Imagine a cluster of input signals. The cluster will act as single input signal when viewed from distance because the direction dispersion is low. On the other hand, input signals will cancel each other when a particle is inside cluster and a stable local equilibrium point is established. It is impossible to create a configuration with no

永久为零。[Corbern94]
equilibrium point due to the additional repulsive effect in closer proximity of an input signal. It is easily provable that any finite number of input signals can not cancel this repulsive effect, therefore the equilibrium point always exists around any input signal. Obviously, many of these equilibrium points are unstable and small, especially when a signal is located at the boundary of the cluster. There is a different situation inside the cluster’s core. The collaborative action of many neighboring input signals stretches the each’s particle zero gradient ring and a smooth and stable local equilibrium is created (fig. 5.2b).

The following picture (fig. 5.3a) shows the force field generated by the previously referenced two-dimensional training set (see sec. 2.6.2). The training set contains five hundred input signals arranged to several clusters. Notice how the force field forms local minima of in each cluster’s center. When a particle is thrown in such force field, it travels along descending gradient until a minimum is reached. A landscape may be imagined instead of the force field with particles as solid balls rolling downhill the surface.

![Figure 5.3: Force field visualization – two dimensional dataset with 500 input signals: The force field generated by input signals is shown on the left side. The effect of repulsive inter-particle force is shown on the right side.](image)

Inter-particle repulsive effect

The inter-particle repulsive force emulates the winner-takes-most behavior of the competitive network. It is crucial to have this element in the system or there will be serious problems with the quality of the final distribution of units. Again, when an surface/balls approximation is used, consider all balls ending in the nearest depression of the landscape. When the inter-particle repulsion is added to the system, the balls can not occupy the same equilibrium point and they can be ejected from already populated local minimum. The figure (fig. 5.3b) shows the change of the force field when the inter-particle repulsive effect is taken into account. The force field trajectory of the particle was recorded during the learning process when all positions are changed concurrently. The trajectory may not reflect the final configuration of the force field correctly.

---

2The trajectory of the particle was recorded during the learning process when all positions are changed concurrently. The trajectory may not reflect the final configuration of the force field correctly.
was generated with respect to the highlighted particle, and therefore, the force generated by this particular unit is omitted. Notice how neurons build a barrier around them to disallow any other neuron to get closer.

When the inter-particle repulsive effect is added to the system, the existence of a stable equilibrium point may be endangered. The repulsive force may be too big and may overpower the attractive force generated by clusters of input signals. Such behavior may create situation where all particles are repulsing each other infinitely and the learning process fails. This situation can be avoided by introduction of an additional condition. If we analyze the definition of the forces (compare eq. 5.8 and eq. 5.9), then setting exponent \( h_1 < 2 \) ensures that the inter-particle repulsive force can not overpower the attractive force of clusters.\(^3\)

5.4.2 Effect of velocity

If one compares the Euler step (eq. 5.6) with the basic Hebbian rule of change of the weight vector (eq. 2.13), the similarity is not as obvious as it could be. The position of the particle in the next step is dependent on the current particle position and the velocity. The velocity of the particle has to be changed directly according to the input signal to obtain the same behavior as defined by the standard competitive learning rule. The velocity is the additional stage inserted between the input signals effect (forces) and the adaptation (position change) and it influences the learning process significantly. However, almost the same approach is used widely in training process of MLP network and it is called the momentum (eq. 2.8). Moreover, there is a momentum defined in classical mechanics as well and the following text shows that the name for the neural-based momentum was not chosen inadequately.

First, there is simple relation between the classical momentum and the velocity: \( p = mv \). If the mass is omitted (set to 1), the momentum practically become velocity. Next, if we have an example configuration of only one neuron which is winning the competition every time and there is only one input signal, the standard adaptation is defined by the following recursive equation:

\[
\begin{align*}
    \mathbf{w}(t+1) &= \mathbf{w}(t) + \eta (\xi - \mathbf{w}(t)) \\
    \mathbf{w}(t) &= \mathbf{w}(0)(1 - \eta)^t + \sum_{i=0}^{t-1} \eta^i (1 - \eta)^i \\
    \mathbf{w}(t) &= (\mathbf{w}(0) - \xi)(1 - \eta)^t + \xi 
\end{align*}
\]  

(5.10)

The last equation tells us that if we have the learning coefficient \( 0 < \eta < 1 \) then in infinite time \( t \to \infty \) the neuron weight \( \mathbf{w} \) becomes \( \xi \). If we analyze the same situation in the physically based approach, we will get a simple harmonic oscillator without dumping:

\[
\mathbf{x}(t) = (\mathbf{x}(0) - \xi)cos(2\pi ft) + \xi 
\]

(5.11)

Due to the conservation of energy, the particle will oscillate the input signal infinitely. If we want to reach the behavior similar to the standard competitive model the damping force need to be introduced to the system. The damping force is velocity (!) dependent and it acts against its direction:

\[
\mathbf{F}_d(p) = -cv_p; \quad c \in \mathbb{R}^+ 
\]

(5.12)

The momentum factor \( \mu \in (0, 1) \) serves as stabilizing element for the neural network training process (eq. 2.8). It improves the ability of unstucking from the local minimum of the error

\(^3\)Precisely, there is a distance where the attraction become bigger the the repulsion.
function by transferring the part of the weight change over several learning steps. If we compare the damping force and the momentum factor from the standard model, it is right to say that without any dumping force \((c = 0)\ i.e.\ no\ force\ acts\ against\ the\ velocity)\ the\ momentum\ factor \(\mu\) is equal to 1 and the whole weight change is stored in velocity, and consequently, it is transferred to the next learning step. The introduction of the dumping force \((c > 0)\) is equal to setting the momentum factor lower than 1. As a conclusion, the velocity element and the damping force serve as the momentum factor similar to the standard machine learning model, therefore there is no problem with our interpretation of velocity. The following figure shows behavior of physical system with and without the damping force (fig. 5.4).

![Figure 5.4: Distance progress with various learning rules](image)

5.4.3 Effect of dimension

The dimension of the input dataset can affect the learning process significantly. The physically based models are mostly tied to two or three-dimensional spaces but the machine learning does not have such constrain and high-dimensional cases have to be considered. For example, the previously used MNIST dataset has patterns of 784 dimensions. There are fundamental differences between the data space of such dimension and the natural three-dimensional space where physically based models operate (see sec. 2.6.7 for details).

The dimensionality problem, if watched from the view of the physically based model, can be understood as a problem of longer distances. The standard competitive method does not need to solve such problem because, in most cases, the learning rule linearly depends on distance, and consequently, longer distances just result in stronger adaptation. On the other hand, the physically driven competition rely on the indirect and nonlinear distance function where overall long distances turn to serious issue. Of course, there is an option of not solving the problem at all and leaving the learning process unchanged which still results in a functional method. Unfortunately, such solution is very time consuming in terms of needed iterations of the learning process, and therefore, we need to speed up the learning process somehow. Generally, there are two possible solution:

Increase in learning rates

All learning rates involved in the learning process may be increased proportionally and the resulting learning process will need less iterations to achieve the sufficient final configuration. However, there can be problem with exact values of increments but practically, the experiments showed that fixed multiplication of all forces constants is highly applicable.
Reduction of distance importance

The importance of the distance can also be reduced by direct multiplication of the distance value in the forces evaluation process. For example the distance increment can be reduced according to the maximal possible distance between data points:

\[ d_i = \frac{\xi_i - x_p}{\sqrt{n}} \]  

Practically, it is useful to combine both methods and after several test runs the optimal learning coefficients can be set. Finally, some non-random weights initialization process can also improve the learning process and suppress the effect of high dimension.

5.5 Physically based self-organizing feature map

The idea of the physically based Self-organized Map is very intuitive and straightforward once the physically based competitive network is established. The model already contains the winner search element and the simple adaptation step that are implemented as forces affecting particles. The only element that is missing is the definition of the neighborhood effect (see eq. 2.29). It is not a surprise that an additional force satisfies this request. The force has to affect not only direct neighbors of the winning unit but also its indirect relatives in longer distance. The spring-based grid constrains between the units of the map has been provided for this purpose. If the relation \( C \subseteq P \times P \) of connections is provided in the same manner as in the standard SOM, then the neighborhood force \( F_s \) affecting particle \( p \) can be defined as follows:

\[ F_s(p) = \sum_{i \in Ud(p)} -kd_i \quad d_i = x_p - x_i ; \quad k \in \mathbb{R}^+ \]  

The spring force is evaluated for all particles in every time step, and consequently, the neighborhood factor is distributed reactively, i.e. parts of adaptation is forwarded to indirect neighbors through spring connections. The inter-particle repulsion (eq. 5.9) and the dumping force (eq. 5.12) act against the spring force, and together, they provide the distance decreasing influence. Examples of physically based SOM are provided in the next section.

The interesting fact is that such reactive model with spring constraints is well known and it is widely used in the area of real physical simulations, like simulations of clothes or simulations of soft bodies [Kaldor08].

5.6 Experiments

This section contains several experiments where the physically based competitive network has been used. The experiments were run above previously used datasets, such as various two-dimensional artificial sets, MNIST dataset, etc.. The goal of these tests is to experimentally show that the physically based competitive network has at least equal learning capabilities as the standard competitive network of the same kind.

---

\(^4\)Of course, such transformation require standardized training set
5.6.1 Quality measuring

The results of various learning methods has to be measured in some way. If the goal of the learning is to quantize the dataset with weight vectors (neurons) then the *distortion error* measure is sufficient enough and the same measure has been already used in the previous experiments. However, I would like to measure another interesting feature of unsupervised learning models with is connected to the random initialization of the learning process.

The initialization of the leaning process may affect the resulting configuration significantly. Almost every machine learning method introduces a kind of random initialization process within its algorithm, and consequently, different final configurations of the system can be expected. This may lead to unpleasant but sometimes very often situations where a user of the algorithm falls down to series of trial and error runs wishing for a lucky hit. Such approach often indicates that the method is not configured properly or can not be applied on particular data. On the other hand, if the learning process converges to similar final configurations regardless of the initial configuration then it indicates that we are approaching the optimum result and/or boundaries of learning capabilities have been faced.

Therefore, it is useful to measure the similarity of resulting configurations after the learning process is finished. There are many methods for measuring similarity and a need for such measure is spread across the whole area of the machine learning. For example, the similarity measure is a critical part of various evolutional methods where such measure is used for the control of the population variability or it is a part of the fitness evaluation process.

There are several suitable methods for measuring similarity between structures of unsupervised neural networks. The structure mainly consists of neurons which are located in particular positions in the input space according to their weight vectors. The task of the point clouds similarity is discussed massively in the area of shape matching and 3D reconstruction/compression [Ankerst99, Hubo07]. However, authors aim on more sophisticated invariants, such as the scale and rotation invariance, which is not needed nor requested in our similarity measure, and therefore, a slightly different method have to be defined. In the matter of fact, the simple nearest neighbor similarity has been found useful for our purposes.

Let there be two competitive networks with two sets \( N_1, N_2 \) of neurons. The simplified algorithm of the NN similarity measure can be described as follows:

1. For every vector \( i \in N_1 \), find its nearest neighbor \( j \in N_2 \) and accumulate the error value \( \| w_i - w_j \| \).
2. Repeat step 1 but switch \( N_1 \) and \( N_2 \).
3. Make the resulting error value proportional to the number of neurons \( m \).

The resulting similarity value can be summarized as follows:

\[
s(N_1, N_2) = \exp \left[ -\frac{\epsilon}{m-1} \left( \sum_{i \in N_1} \arg \min_{j \in N_2} \| w_i - w_j \| + \sum_{i \in N_2} \arg \min_{j \in N_1} \| w_i - w_j \| \right) \right] \tag{5.15}
\]

The coefficient \( \epsilon \in (0, 1) \) denotes the rate of similarity where values near 1 result in very restrictive measure and vice versa.
Finally, it is important to point out that such measure is not truly ideal because there are situations where the measure incorrectly rank possibly similar final configurations. The real problem arise if there are clusters with uniformly distributed members. In such situations, the location based measure may penalize these clusters (i.e. units within these clusters) because there are many distributions of units which meet the minimum distortion error condition. Therefore, the similarity measure should be used along with the basic distortion error measure.

### 5.6.2 Two dimensional dataset

The following example shows the previously used two-dimensional dataset. Notice the trajectories recorded during two independent learning process. The trajectories are different due to the different initial positions of particles but final configurations are spatially equal. Moreover, the ”dead unit” is located at the same position as well.

![Figure 5.5: Physically based self organization – 2D dataset.](image)

The next figure (fig. 5.6) contains the plot of error values gathered during repetitive learning processes that have been run on this particular dataset. The goal of this experiment was to measure the similarity of final configurations. Appropriate distortion error values are plotted on the Y-axis. Two corresponding models were chosen, the standard hard competitive learning method (see sec. 2.6.2) and previously described physically based competitive network. Additionally, the standard GNG model has been tested against the same dataset. The learning configuration of all used models have been set in order to achieve the same final number of neurons (9 neurons). The similarity measure (eq. 5.15) has been applied and all possible pairs of cases were compared. A pair has been designated equal if the similarity reached the value of 0.95 within the rate of similarity $\epsilon = 1$. Equal configurations are highlighted by different colors.

The convergence ability of the physically based competitive network is clearly visible. Precisely, 66% of training cases led to the optimal configuration where the standard hard competition failed to achieve such result; only 12% of training cases hit the optimal configuration. In comparison, the GNG method reached the similar result of 64% of optimal cases. Moreover, 25% of physically based competition cases converged to the second best optimum. On the other hand, it need to be highlighted that the physically based training might get stuck in a very poor local minimum; in fact, the worst solutions have been produces by the physical competition.
Figure 5.6: Similarity measure – physically based learning; 2D dataset. The learning process was repeated 50 times for every model. Resulting error values are drawn into the graph. Also, the similarity measure was evaluated among all cases of each model. Equal configurations are highlighted.

Physically based SOM

The next experiment uses the same dataset but the model has been changed to the self-organizing map and its physically based opposite. The following pictures (fig. 5.7) show results of the training.

![Physically based SOM 1](image1)
![Physically based SOM 2](image2)
![Standard SOM](image3)

Figure 5.7: Physically based SOM – 2D dataset. Two different configurations with different spring factor were uses for the physically based SOM.

Evidently, the physically based SOM produces very similar structure as the standard SOM algorithm. However, the standard SOM method is slightly better in the meaning of achieved distortion error but the difference is not very large; practically, the physical SOM can be tuned to follow more similar distribution (compare two different physical SOM learning cases – fig. 5.7a vs. fig. 5.7b).

Again, the similarity measure was used to test method’s convergence and to test dependency on the initial distribution of neurons. Results are summarized in the following plot (fig. 5.8). The physically based learning demonstrated slightly worse results but still, they are comparable to the standard approach. Unfortunately, some poor result may appear while the physically based SOM is used but on the other side, the probability of such case is very low.
Another interesting fact about the physically based competition is the progression of the distortion error value over the learning process. While the standard competition produces strictly decreasing curve, the physically based learning may result in quite unbalanced progress, but finally, a minimum is approached as well. The next figure (fig. 5.9) shows progress of the error value during the learning process.

The standard competitive learning rule is based on the subtraction between the current neuron position and the input signal passed to the network (see eq. 2.13) and such learning rule disallows the uphill movement of the overall weight change. On the other hand, the physically based competition involves the momentum (see sec. 5.4.2) which conserves energy of particles and allows oscillations around local minimum. Such behavior may lead to better convergence in a term of finding deeper minimum while the neuron is oscillating around the current location and the same approach is widely used within the area of machine learning.

5.6.3 Iris plants dataset

The purpose of this experiment is to measure the similarity between various final configurations of the physically based competitive network. The well-known iris plants dataset [Frank10] has been used for this experiment. The dataset contains 150 instances of plants where all four attributes are physical dimensions of the plant’s floret. The next graph plot presents the
distortion error values gathered during experiment. The similarity has been measured according to the following attributes: the similarity rate $\epsilon$ has been set to 1 and a pair of final configurations has been considered equal if the similarity reached the value of 0.95.

![Figure 5.10: Similarity measure – Iris dataset. The learning process was repeated 50 times for every model; both models contains 10 neurons. Resulting error values are drawn into the graph. Also, the similarity measure has been evaluated among all cases of each model. Equal configurations are highlighted.](image)

Again, it is clear that the physically based competition is not much better in a term of the best minimal error value\(^5\) but the average result is definitely better. Precisely, 68% of final distributions produced by the physically based learning were equal against the similarity measure. Moreover, if we harden the measure of quality to the value of 0.99, i.e. the average distance to the nearest neuron position must not be greater then 0.01, then 50% of final distributions will remain equal.

![Figure 5.11: Similarity measure – SOMs; Iris dataset](image)

**Physically based SOM**

The same experiment can be executed above the physically based SOM and the similarity of resulting grid structures within the input space of the dataset can be measured as well. The results of the physically based SOM and the standard SOM were almost equal although the standard SOM reached slightly better distortion error level (fig. 5.11). The learning process has

\(^5\)In fact, it is impossible to achieve better result due to the number of neurons.
been repeated 50 times for every method and the as a result, 90% of final distributions were equal for the physically based SOM; the standard SOM reached 98% of equal final distributions. A positive fact has been found: no more poor results with high error value appeared unlike in the 2D dataset experiment above.

5.6.4 MNIST dataset

Experiments in this section are provided in order to examine the impact of high-dimensional data to the physically based competition. The dataset used in the experiments is again the MNIST dataset of hand-written numbers [MNIST98]. First, the simple physically based competition has been tested. The following figure (fig. 5.12) shows an example network with 20 particles after the equilibrium state has been reached. The network neurons (particles) are visualized naturally as $28 \times 28$ rasters. The images are not ordered and should not be ordered in any way due to the absence of connections or other relations between particles (neurons).

![Figure 5.12: Physically based competition – MNIST dataset](image)

It is clearly visible that the physically based competitive network is capable to distribute particles according the distribution of the original input dataset. All contained digits have been "found" and further experiments showed that the network reached approximately equal error value as the reference GNG network model run above the same dataset. However, the learning is not ideal; notice the highlighted dead unit which is not located in any digit subspace.

**Physically based SOM**

Again, the dataset has been tested against the physically based SOM i.e. against the particle system with spring connections. The following figures (fig. 5.13) show two different final configurations of the physically based SOM after the equilibrium state has been reached.

The configurations differ in strength of the spring connections. The figure on the left side was produced by the physical SOM with weak springs and the figure on the right side was generated by the SOM with stronger spring connections. Finally, it is clear that the neighborhood can be simulated by spring connections and it remains functional regardless to the dimension of input data.

5.7 Improvements to standard competitive learning

Experiments presented in the previous section showed that the physically based competitive network is functional and the quality of the resulting network structure and consequent distortion error levels are equivalent to the standard competitive network. In this section, I would like to summarize possible improvements that the physically based approach can provide in the area of unsupervised learning.
5.7.1 Reduction of random initialization effect

Presented experiments showed that the physically based approach provide more stable convergence due to the different learning mechanism where a kind of momentum factor is involved; in fact, is it the real momentum quantity as it is known from the classical mechanics. Such improvement can have positive practical impact to a wide area of experiments. If the random initialization has less influence to the quality of the learning process, then one need not to repeat an experiment several times waiting for the ”good” starting point.

5.7.2 Breakable Self-organized Map

Another possible improvement lies in the modification of the physically based SOM. It is possible to follow the physical nature of the method and extend it by destructible connections between network units. Such behavior may be implemented as maximal allowed force condition which should be tested in every simulation step i.e. iteration of the algorithm. If the spring force exceeds the allowed value then it is assumed that the spring can not sustain such tension and tears up. The idea has been successfully tested against the previously used two-dimensional dataset and the following figure shows the resulting network after the learning process reached the equilibrium state.

The tearing ability of the physically based SOM can help the method to acquire better final configuration in a term of lower distortion error. In other words, if the network can not break apart, then there is a possibility of existence of units which lie in areas where no input signals are present. Such behavior wastes resources (available neurons), and therefore, it should be avoided if possible. Moreover, the broken connections can serve as natural boundaries for the clustering mechanism and can replace the standard \textit{U-matrix} based clustering.

However, there are some problems with this modification. First, the random initialization of weight vectors can result in situation where the spring force is higher then allowed threshold. This results in undesired degradation of the grid before the map can even start to self-organize, and therefore, some additional modification is needed. Possible solution to this problem is to introduce the global “age” for connections in a term of decreasing of the tearing threshold over time. At the beginning, all connections are “young” and it is not possible to break them.
As the algorithm proceeds, connections become “older” and the possibility of break increase. Practically, such improvement is mandatory and has to be implemented if a functional breakable map is requested.

Second, it should be noted that the breakable map is not always the best solution due to the different final distribution of neurons. Sometimes, it may be useful to have neurons that cover the void between clusters of input signal, especially when the network is intended for classification. Usually, there is no guarantee that the training set covers all possible areas where input signals may come from and additional neurons distributed between these areas can improve the ongoing classification process. For such reason, the usage of the breakable map is highly application specific.

5.7.3 Implementation notes

Nowadays, the parallel or distributed computation is very popular in the area of computer science. The computational power of the newest hardware can not be easily enlarged without the parallel approach any more, and consequently, parallel forms of many computation methods and algorithms are sought and designed. Moreover, there is an area of specialized GPU [parallel computing which evolves very rapidly and provides powerful hardware resources but has very specific needs and usage conditions. In this section, I would like to suggest a GPU driven implementation of the physically based competitive network and make some estimations about the possible speedup of such approach.

The GPU based implementation of the standard competitive learning has been addressed several times using different techniques. From the historical point of view, the GPUs are mainly designed to process graphical information like textures, vertexes, and other CG specific data due to their origin in the video game industry where the main purpose was to speed up the evaluation process of the graphical pipeline. If such hardware is used for more general computational purposes, the algorithm has to be altered to fit into boundaries of the graphical pipeline evaluation [Zhongwen05, Xiao10].

Fortunately, the GPU market is now turning to the parallel computation and there are frameworks which provide more general middleware between the GPU hardware and the developer.\footnote{Graphical Processing Unit}
Probably the most used framework nowadays is the NVIDIA CUDA\textsuperscript{7} framework \cite{Nvidia10}. This framework supply the programmer with very sophisticated tools for creating computational threads, memory blocks etc. in order to execute the code on graphical hardware.

There are implementations of the standard SOM for the CUDA environment \cite{Sijo08} and we can start by comparing the possible implementation of physically based SOM to this reference CUDA implementation of the standard SOM. As it was mentioned above, there are some attributes and conditions which have to be considered if the GPU driven implementation is prepared. First, the algorithm has to be designed to provide as much parallelism as possible. Such condition is obvious and it doest not change regardless to used hardware (multi-CPU, GPU etc.). Next, memory management and flow control have to be redesigned to get best possible performance. The memory management is tied to the issue of transporting input data to the video memory as effectively as possible with respect the the memory structure of the CUDA device. Since input data remains the same for both methods, this task can be solved in the same way and there will be no performance differences between them. On the other side, the flow control will differ significantly. The nature of the SIMT\textsuperscript{8} architecture of the CUDA device operates with the mechanism of serialization of all instruction branches which produces significant performance loss when almost any conditional code is executed within the same warp\textsuperscript{9}. If we analyze the standard SOM algorithm then we may found several points where the algorithm needs to be synchronized and some conditions need to be solved. The step of the standard SOM algorithm can be briefly summarized as follows:

1. Fetch input signal.

2. Search for the nearest unit – the winner:
   (a) Evaluate distance to every neuron.
   (b) Find minimal distance.

3. Update weights according to the winner position in the grid.

Clearly, step 3 of the algorithm depends on the second step. Additionally, there are only limited possibilities for how step 2b can be parallelized, and consequently, it is not possible to fully parallelize the whole algorithm at once. However, practical experiments showed that even the partial parallelization of the SOM algorithm can gain massive speedup when compared to the standard CPU implementation \cite{Sijo08}.

On the other side, the step of physically based SOM processes all input data at once as it simulates the single integration step of the physical system.

1. Evaluate all distance dependent forces according to positions of particles and inputs signals.

2. Update all primary physical quantities (velocity, momentum, position) of all particles.

The forces definitions are condition-less and there is no winner search nor any minimum search, and therefore, the whole step of the algorithm can be parallelized. In fact, the physically based learning process is a special case of the \textit{n}-body problem \cite{Diacu96} which has a

\textsuperscript{7}Compute Unified Device Architecture
\textsuperscript{8}Single Instruction Multi Thread
\textsuperscript{9}Briefly, a group of threads starting at the same point in the algorithm
very successful CUDA implementation [Nyland07]. The reference implementation can perform around 10 billion body-to-body interactions per second when the three-dimensional system of 4096 particles is simulated which roughly yield in 1000 simulation steps per second. The real physical SOM implementation needs to be modified to support any-dimensional task and such modification will hit the overall performance of the algorithm. However, the physical SOM is based on the particle system where only a few particles (neurons) can change their positions and the rest of particles (input signals) are only sources of gravity. Therefore, many evaluations of position and velocity changes can be omitted and the algorithm can be sped up.

As a result, the CUDA implementation of the physical SOM will be incomparably faster then the CPU implementation and it should be faster then the CUDA based standard SOM implementation. My future work in this topic will be focused to the implementation of such system and real performance results have to be measured as well.
Chapter 6

Conclusion

In this dissertation thesis, I have summarized the majority of my contribution to the competitive networks topic which I have worked on during my doctoral studies. I tried to examine the topic from different angles, and also, a lot of ideas came from specific applications of neural networks which I have faced. My personal interests in computer graphics and visualization methods lead me to express my results mostly in a graphical manner and it also drove me to think of the ways in which the competitive network can be presented visually and if there are any positive attributes of such approach.

I am aware that my thesis might be considered a little unfocused or broad in its range of discussed problems, but it clearly came from the fact that I tried to understand the competitive network and tasks of competitive learning in different ways and find some other possible interpretations and applications of the well known principle of competition driven learning. Nevertheless, I did not forget the scientific method; I tried to provide necessary experiments to all proposed improvements and ideas to prove that my suggestions might have a practical impact on the topic.

The following text is a summary of the results I have achieved in the topic of competitive networks. The summary is divided into three sections.

First, an improved competitive network is summarized. The improvement lies in the structure of the network and different understanding of supervised competitive learning; therefore, the model is intended for the task of classification.

Second, a possible application of this network within the problem of weight initialization of the standard feed-forward network with radial basis function units is resumed along with another uncommon application of competitive network which is focused on the visualization of high-dimensional datasets and where the competitive network is merged with the physically based, graph plotting method.

Finally, a new insight into the paradigm of the competitive network is recapitulated. The competitive network is threaded as a pure particle system based on physical laws of motion. Various similarities and differences are pointed out and it is shown that such model is at least equal to the standard model and might have some valuable attributes such as lower sensitivity to the initial random configuration.

6.1 Improved classification

An innovative approach to the classification task done by competitive network has been presented. The innovation lies in joining the input data with the desired class information. Usually, the supervised training uses the input data for the adaptation of network weights but desired value (or values) are used in terms of correcting such adaptation process. The joined approach uses all input and desired output data together as a single joined input vector. The desired output has to be a one-of-C coded class vector but this is very common for classification purposes.
Such joined vector is passed to the network in the learning stage and no supervision is needed. A classifier based on the standard competitive network needs to have additional information stored in neurons. Such information can be set a priori by coloring all neurons with a single class value. The adaptation process is then supervised according to these values. Another common way is to store all desired class values mapped on each neuron during the training phase and at the end, all neurons can be assigned with the most mapped class value. Then, it is very easy to build a classifier when an unknown input signal is provided to the network. The winner of the competition can be found in the standard way and the class information stored in the weight vector of the winner unit classifies the input.

The situation is different when the joined competitive network is used. The standard competitive network has weight vectors with the same dimensionality as vectors from the input dataset, and therefore, the winner can be found as usual. Obviously, the weight vectors of the joined competitive network are extended by the class information, and consequently, the winner of the competition can not be evaluated directly. However, if the class information stored in extended weight vectors is omitted, the winner can be found in the standard way. Finally, the omitted class information of the winner serves as the output of the network.

One may ask why to bother with the joined input vector approach instead of the standard supervised competition. There are basically two reasons to do it. First, when a classifier is built upon the standard competitive network, the algorithm has to be extended with a supervision/correction mechanism or a method for coloring the neurons with class marks has to be provided. On the other hand, the joined competitive network does not need any of these. Simply, all available information is treated as the input of the competition. The output of the network is evaluated by simple alternative omission of parts of the extended weight vector.

The second reason to use the joined competition is that the classic competitive classifier can not perform soft classifications effectively. The output of the network is produced according to the color mark of the neuron which is a single class value, and so no soft classification is possible. Alternatively, it may seem that incrementally built class mapping can produce soft-decisioning. It is true that soft classification can be evaluated according to the number of different classes mapped on the same neuron during the learning process but there are problems with this approach, especially when a growing network is involved. If the mapping is constructed during the whole learning process, then at the beginning of the learning, the initial neurons can gain undesired mapping from almost all patterns from the training set. Of course, the common way to avoid such behavior is to limit the mapping creation to the last iteration through the training set but even then the resulting soft classification can become too fuzzy in certain cases.

The joined competitive network produces soft classification naturally. Additionally, I have provided a prove that the output of the network fulfills the condition of probabilistic classification without any correction (see sec. 3.2.4). Moreover, the extended weight space gives the network more freedom in the learning phase and the final distribution of neurons can be much better than in the standard case, especially on boundaries between different classes of inputs (see fig. 3.1 and fig. 4.3). Such outcome has been experimentally proven on several datasets (see sec. 3.4).

---

1 In comparison with common supervised methods like LVQ.
2 Again, the class information has to be in the one-of-C form.
3 One may think of analysis of second winner, third winner etc. and using a kind of interpolation between them but such modification basically pollutes the algorithm with another possibly defective measure.
4 The sum of all values is approximately 1.
The MNIST experiment shows the classification on real data with high dimension\(^5\) and it is very clear that my assumptions about the soft classification were correct. All experiments were done with the joined modification of the known Growing Neural Gas model developed by B. Fritzke.

### 6.2 Competitive network as supporting mechanism

There are many possible applications of competitive networks. In my thesis, I have presented a few of them that have close connections to the topics I am interested in.

#### 6.2.1 Initialization of RBF network

One of the known and useful applications of the competitive network is the weight initialization of the RBF network. The RBF network contains so-called local neurons with a kind of Radial Basis Function as their threshold function. For clarity reasons, I have proposed a quite straightforward deduction for how the competitive learning rule naturally appears from the common error-based gradient descent method when the RBF units are present in the network structure (sec. 4.1.2).

Next, I have dealt with a possible application of the Joined Growing Neural Gas method introduced in the previous chapter. I have shown that the boundary preserving behavior of the joined input approach can result in better distribution of the RBF layer and can speed up the ongoing backpropagation learning process of the output layer (sec. 4.1.4). Moreover, I have suggested additional initialization of the output layer of the RBF network (sec. 4.1.5). I have applied the idea that the output of the joined network is not only the distribution of units, but that there is also the output classification vector which can be used as a template for output units of the RBF network. Finally, several experiments have been provided to support the premise that such initialization steps will speed up the fine tuning driven by backpropagation.

#### 6.2.2 Visualization of high-dimensional data

Another application of a selected competitive network (the GNG network) has been presented. This application was intended to be a tool for a high-dimensional dataset analysis. First, I described some common practices in visualization and I pointed out the importance of visualization. I also run through the most used data visualization methods including competitive ones, like the Self-organizing Map.

Next, I have contributed to the topic by introducing the method which utilizes the Growing Neural Gas network and a physically based graph plotting algorithm. As a result, a fully applicable tool for insight into the high-dimensional dataset has been constructed. Some attractive attributes of the merge of competitive learning and physically based layout has been pointed out; for example, the positive influence of concurrent runs of the force based layout algorithm and the competitive learning which results in better convergence of the relaxation. Finally, several dataset visualizations have been presented to support the idea of this visualization method and some other possible improvements have been suggested.

---

\(^5\) A 28 \( \times \) 28 raster plus an additional 10-dimensional class vector gives us 794-dimensional input vector.
6.3 Physically based competition – A different insight

My research in the previous topic (the high-dimensional visualization tool) can also serve as a link to the last chapter of my thesis. During the work on this research I realized that there is a close connection between the relaxation process of a physically based particle system and competitive learning in general. I tried to elaborate this idea and I realized that it is possible to build a physical model which has the very same behavior as the competitive network.

I have summarized this insight in my thesis along with detailed explanations of various features of such system. First, a formal definition of the physically based particle system has been provided. Next, the structure of a physically based competitive model has been presented. Several dual attributes have been pointed out, such as the learning rules of competitive network and the definitions of forces in the particle system.

The learning process has been compared to the relaxation process of the particle system and other similarities have been highlighted; for example, the relation between the well known momentum factor from machine learning methods and the velocity based dumping force stabilization, which is widely used within various physical systems. Furthermore, some rather theoretical issues have been addressed such as the effect of the dataset dimension or the existence of the equilibrium state of the particle system which is treated as the final configuration of the trained network.

As I found the basic model to be successful, I have suggested another model inspired by the famous Self-organizing Map. The physically based model of the Self-organizing Map has been presented where the neighborhood factor of the standard model is altered into the set of flexible spring-based connection between particles.

For demonstration purposes, several datasets have been tested against the physically based competitive network including the previously used two-dimensional dataset, iris plant dataset, MNIST dataset, etc. It has been shown that the model expresses itself with the same basic behavior as the standard competitive network and that the assumption about the equilibrium state as the desired final configuration was right. Moreover, some interesting attributes have been found. It became clear that the physically based competitive network can overcome some known issues of the standard model such as the problem of unbalanced clusters of units or the occurrence of dead units. These attributes are closely related to the "force landscape" generated by the input signals and other units of the network, and therefore, it is crucial to precisely set all learning factors (i.e. forces constants).

Several experiments have been run for the physically based Self-organizing Map as well. It has been shown that introduced spring connections can serve as the neighborhood preserving mechanism which results in a behavior similar what was predicted. Moreover, it has been shown how the spring configuration can affect the resulting network grid structure. Some additional abilities of the physically based SOM model have been highlighted. For example, it is possible to set the maximal allowed tension for spring connections. Such condition will result in the breakable grid which gives the model more freedom to occupy the given data space. In such manner, even a very sparse dataset can be organized without useless and undesired inter-connection units.

Finally, some implementation notes have been given. It has been pointed out that the physically based approach can be effectively distributed to a large number of computational units without any flaws that mar the standard approach. In fact, the algorithm of evaluation of a network unit weight change is totally independent to other units (there is neither winner
search nor the distance dependent sort), and therefore, it is possible to accompany as many computational streams as the number of network units. Additionally, a CUDA implementation is suggested, due to the natural condition-free structure of the algorithm.

6.4 Future work

In the future, I would like to focus on the application part of ideas I have proposed in my thesis. First of all, I would like to finish the CUDA/GPU based implementation of the physical competitive network and provide real performance tests of such system. I am convinced that the GPU implementation will provide valuable speed up to the system and improve the usability.

Next, I would like to extend the family of physically based competitive networks by other models including the GNG method or other growing networks. In fact, a physically based solution to the growing ability is still missing but there are some very interesting possibilities for how such problem can be solved e.g. the introduction of mass and ongoing disintegration of parent particles.

Moreover, it would be interesting to combine the physical competition with the joined dataset approach I have suggested and a classifier based on this mixture may be built. However, proper experiments have to be provided to test usability of such model.

The visualization of high-dimensional datasets is another topic I am interested in. The proposed visualization method can be extended in various ways. For example, the immediate visualization of weights inside the artificial neural network during the training process might be very useful and can provide valuable information about the quality of training. Nevertheless, the visualization itself can help in the process of validation of any new machine learning model and I have found it extraordinary useful.
Chapter 7

Bibliography


Chapter 8

Appendix

8.1 ANN ToolBox – Description of software

A short description of my software tool will be given in this section. The ANN ToolBox is the software which has been created for experiments with various artificial neural networks. I’m aware that there are many sophisticated tool already implemented but I decided to create another one by my self for several specific reasons:

First, It is crucial to known the exact functionality of many learning methods and the best way for how this can be achieved is to implement them on your own. Second, the improvements and innovative approaches I have suggested has to be experimentally tested and a functional framework suited for such extensions is very helpful. Finally, I used to utilize various visualization techniques within the work on my thesis, and therefore, some very specific graphical output was needed. Unfortunately, there are not many neural networks tools which have such features.

The graphical user interface has been prepared to provide as much information as possible through one screen, and therefore, the application is created by a number of sub windows which can visualize various aspects of current neural network. The position and size of each sub window can be customized and the whole application window layout can be stored for later use. In such way, different layouts can be prepared and reused for different needs when various neural networks are processed.

The application incorporates many useful visualizations and tools and some of them are shown on the following figure of the main application window (fig. 8.1):

1. The network and the training dataset can be loaded through toolbar or application menu. The learning process can be managed here as well.
2. Every implemented algorithm can be easily configured through the property editor.
3. The state of the network can be inspected through the network structure window which is updated repetitively during the learning process. Various colors are used to describe neuron weights, thresholds etc..
4. The progress of the learning process can be viewed by plots of time dependent values, e.g. the error of the network trainer.
5. Simple projections of input data can be plotted into the network output map window. A the name of the window denotes, the output of the network within the selected domain is visualized as well.
6. The textual summary of the learning algorithm is also provided.
7. All executed operations and other information about the current state, for example some statistical data about the finished learning, are added frequently to the console window.
Figure 8.1: ANN ToolBox – Screenshot of application main window

The Java SDK has been selected for implementation along with other useful libraries, such as Swing, Java 3D Vecmath, Apache Batik, InfoNode Docking Windows etc.. These libraries allow me to build a very flexible application within a relatively short period of time. However, the performance of such a system can be argued. Honestly, the Java environment is not known for its speed and performance capabilities. On the other hand, the purpose of this work is not to provide the final product for using specific neural networks but it is rather focused on extendibility and easy development where the Java SDK excels by any means. Nevertheless, I have tried to optimize the code in order to achieve better performance results, and moreover, I have provided a module which can incorporate a native code execution if an irresolvable performance flaw appears.

The application has been designed as an extensible framework. There are several class
hierarchies which encapsulate different aspects of the system such as abstractions of neural structures, learning strategies or dataset handling. Much effort has been put to preparation of the framework for agile development, and therefore, it is very easy to extend existing system in order to incorporate a new ANN model. For example, the configuration for a new learning algorithm can be described as a set of annotated class attributes. The resulting class is then processed by the Java reflection library and a property editor is generated automatically. Moreover, the framework is equipped with a sophisticated graphical module which has been built above the standard Swing Graphics API, and optionally, all visualizations can be exported as SVG files. In such way, the developer can focus on the algorithm itself and/or visualization of results and need not to bother with background work which is handled by the framework.

The following list contains some of already implemented artificial neural network models:

**MLP Network** One of the very basic methods. The backpropagation learning and evolitional learning has been implemented as well.

**Recurrent MLP Network** The model capable of maintenance of inner state. Very useful for time series prediction tasks.

**RBF Network** The layered model which incorporate RBF units. Slightly different threshold based transfer function has been also provided (see eq. 4.2).

**Hard Competitive Learning** Basic competitive network.

**Self-organizing Feature Map** Very well-known competitive model also known as Kohonen map. Various neighborhood schemes has been implemented as well.

**Neural Gas** Connection-less type of competitive network where the winner-takes-most adaptation is used.

**Growing Neural Gas/Joined Growing Neural Gas** Extension of Neural Gas model. An innovative joined dataset driven learning has been implemented.

**Learning Vector Quantization** This well-known competitive model has been implemented for comparison purposes.

**Physically Based Competitive Network** A physically based form of Hard Competitive Learning. A few integration techniques has been built (Euler, RK4).

**Physically Based SOM** A physically based form of the standard Self-organizing Feature Map.

**FBA visualization of GNG** A visualization method used for analysis of high-dimensional data which combines the GNG method and the force-based layout algorithm.

The ANN ToolBox application is not finished yet, and probably, it will never be finished. My experiences thought me that it is a very useful data analysis tool with good visualization capabilities. The future work I am planning for this framework is to generalize it beyond the area of neural networks so the that the framework can be used for various global optimization methods or other machine learning topics.

---

1In fact, almost all figures of resulting network structures and experiments in this thesis has been generated in this way.
8.2 Common properties and notational conventions

List of symbols
\( \eta \) General learning factor
\( \lambda \) Steepness factor of sigmoid transfer function
\( \varphi \) Transfer function
\( \rho \) Plane in weight space formed by weight vector of neuron and it’s threshold
\( \theta \) Threshold level of neuron
\( \xi \) Input signal
\( a \) Acceleration of particle
\( 0 \) Zero matrix
\( C \) Set of connections between neurons
\( c \) Binary class vector
\( d \) Desired output vector / subtraction between particle position vectors
\( d \) Distance function: \( d \) – Euclidean, \( d_1 \) – Manhattan, \( d_\infty \) – Chebyshev and \( d_H \) – hexagonal
\( E \) Error value
\( F \) Set of forces
\( F \) Force that affect particle
\( I \) Identity matrix
\( j \) Joined input vector
\( m \) number of neurons in layer / particle’s mass
\( N \) Set of neurons within network or layer
\( n \) Dimension of dataset, input space, etc.
\( P \) Set of particles
\( p \) Training or testing pattern
\( PS \) Particle system
\( S \) Set of springs between particles
\( T \) Set of training patterns
\( t \) Discrete time variable
\( U(u) \) Set of neighboring units to the unit \( u \)
\( v \) Velocity of particle
\( w \) Weight vector of neuron
\( w \) Winner unit / winner selection function
\( x \) Input vector of network or neuron / position of particle
\( \| x \|_1 \) Manhattan norm
\( \| x \| \) Euclidean norm
\( \| x \|_\infty \) Maximum norm
\( y \) Output vector of network
\( y \) Output value of neuron
\( z \) Potential level of neuron
List of terms

batch learning
A variant of a learning algorithm where weight changes (adaptation) is accumulated over a period of time (i.e. a number of iteration) and are applied at the end of this period, for example, after all training pattern have been processed.

Chebyshev distance
A distance measure introduced by the maximum norm. The maximum norm is a $p$-norm where $p = \infty$:

$$d_\infty(q, r) = \| q - r \|_\infty = \lim_{k \to \infty} \left( \sum_{i=0}^{n} |q_i - r_i|^k \right)^{1/k} = \max\{|q_0 - r_0|, \ldots, |q_n - r_n|\}$$

code-book vector
Weight vectors of unsupervised neural network neurons. The code-book vector name is usually used when the number of neurons is fixed and the network does not grow during the learning process.

competitive network
An artificial neural network where the competition of neurons is involved in some manner. Usually, the competition is metric driven. The competitive network has its own biological inspiration, so-called lateral inhibition.

cosine distance
A cosine distance (cosine metric, cosine similarity) is a function between two vectors based on the angle between them. It can be used as a core for the winner search substep of the unsupervised learning algorithm:

$$d(q, r) = \frac{q \cdot r}{\|q\| \cdot \|r\|}$$

Delaunay triangulation
A triangulation $DP(P)$ of set of points $P$ in plane such that no point is inside the circumcircle of any triangle in $DP(P)$. Delaunay triangulation maximize minimum angles of the triangles in $DP(P)$.

distortion error
Distortion or quantization error is a function of an unsupervised network (represented by a set of weight vectors) and appropriate input dataset. A common goal of unsupervised learning is to minimize this function:

$$E(N, T) = \frac{1}{|T|} \sum_{\xi \in T} \arg\min_{u \in N} \| \xi - w_u \|^2$$

Euclidean distance
A distance measure introduced by the Euclidean norm:

$$d(q, r) = \| q - r \| = \sqrt{\sum_{i=0}^{n} (q_i - r_i)^2}$$

fully connected network
A neural network which is created by layers of neurons. There are no lateral connections and all neuron outputs from current layer with $m$ neurons are forwarded to all neurons in
the following layer with \(n\) neurons. Consequently, there are \(m \times n\) connections between layers and neurons in the following layer have \(m\) inputs.

**hexagonal distance**
A distance between center points of hexagons within hexagonal space i.e. the hexagonal tiling of the Euclidean plane. In fact, the distance is the shortest path between two vertices of graph formed by triangular tiling of the same plane which is a dual tiling to the hexagonal one. If we introduce a coordinate system where the \(X\) axis points south-east and the \(Y\) axis points north-east (see fig. 2.14), then the hexagonal distance can be defined as follows:

\[
d_H(q, r) = \begin{cases} 
\|q - r\|_1 & \text{if } (q_x - r_x)(q_y - r_y) < 0 \\
\|q - r\|_\infty & \text{if } (q_x - r_x)(q_y - r_y) \geq 0 
\end{cases}
\]

**incremental learning**
A variant of a learning algorithm where weight changes (adaptation) are applied in every learning step.

**input space**
A vector space from which input vectors are constructed. Usually, it is the Euclidean vector space \(E^n\) of arbitrary dimension.

**input vector**
A vector of input values. We can distinguish input vector of the network and input vector of the neuron. Usually, when the first layer of the network is fully connected then these vectors have equal number of elements.

**labeled data**
A set where every record is labeled with some classification mark e.g. a one-of-\(C\) coded binary vector. Usually, such dataset forms the training set for supervised or semi-supervised training process.

**lateral inhibition**
A mechanism where lateral neighbors of neuron in the same layer are affected by its own excitation. Competitive networks usually used similar behavior in order to adapt neighboring units of excited neuron.

**learning factor / learning coefficient**
Usually, a real number or a set of real numbers which affect the strength of adaptation within the learning process.

**learning process**
An algorithm which repetitively runs similar steps in order to adapt the state of the neural network and achieve the specified goal (e.g. minimal mean error).

**linear separation**
An ability of dividing the input space into two subspaces by linear structure i.e. hyperplane.

**Manhattan distance**
A distance measure introduced by the Manhattan norm also known as the taxicab distance. The Manhattan norm is a \(p\)-norm where \(p = 1\): 

\[
d_1(q, r) = \|q - r\|_1 = \sum_{i=0}^{n} |q_i - r_i|
\]
momentum
A quantity which affect the learning process. There are various learning algorithms which are based on gradient descent or they generally search the weight space in order to achieve goals. The movement inside the weight space can be understood as classical mechanical movement where velocity and momentum is involved. From such point of view, the movement can resist short-time changes by the presence of momentum.

network output
A vector created by output values of all neurons in the last layer of the network.

one-of-C coding
A coding technique used for the classification of C unordered categories. When a one-of-C coding is applied, desired values attached to every training pattern form a C-dimensional binary vector where only one of its elements is set to 1, i.e. it denotes a target category.

pattern
A pair of input signal and vector of desired values: \( p = (\xi, \mathbf{d}) \). Sometimes, if the one-of-C coding is used, desired values form a class vector: \( p = (\xi, \mathbf{c}) \)

potential / inner potential
Weighted sum of all neuron inputs forms inner potential of the neuron which is then transformed to the output value.

probabilistic classification
An ability of the classifier as it produces probability estimate for predicted categories, i.e. the sum of all output values is 1.

reference vector
A weight vector of unsupervised network neuron. May be also called as a code-book vector or a prototype vector.

self-organization
An unsupervised machine learning process where the structure of the network evolves according the the distribution of input signals fetched from the training set.

semi-supervised learning
A machine learning method based on the self-organization principle but the teacher is also involved. The teacher usually corrects potentially undesired steps of the learning process.

soft-decision
An ability of the neural network driven classifier. Soft-decisioning classifier provides real values rather then binary values in order to fuzzy boundaries between predicted categories. Usually, the probabilistic classification is also demanded.

soft-max function
An activation function; may be also called a multiply logistic function. Usually used to enforce that the network will provide the probability estimate as its output. Let there are \( c \) categories, i.e. the network will have \( c \) output units and will provide output \( \mathbf{y} = (y_1, \ldots, y_c) \). Then the transformed probability output \( \mathbf{p} = (p_1, \ldots, p_c) \) is defined as follows:

\[
p_i = \frac{e^{y_i}}{\sum_{j=1}^{c} e^{y_j}}
\]

supervised learning
A machine learning method where the teacher is involved. Usually, the task of the teacher
is to correct the learning process and provide the information of what is right and what is wrong.

**testing set**
A set of patterns used for the testing phase of the learning process. Usually, the intersection of the testing set and the training set is empty.

**threshold**
A value which controls the excitation of a neuron. When the inner potential exceed the threshold value, neuron should be excited, i.e. it generates higher output signal.

**training set**
A set of patterns used for the training phase of the learning process. Usually, the intersection of the testing set and the training set is empty.

**training step**
A step in the learning algorithm which is usually repeated many times and where a weight change substep is involved.

**transfer function**
A function which transforms the inner potential of the neuron to its output value. Usually, it is tied to the threshold value.

**U-matrix**
The U-matrix (unified distance matrix) is a quantitative used for visualization of neighborhood within Self-organizing Feature Map (SOM) where the Euclidean distance between neighboring neurons is evaluated and visualized as a gray scale image.

**unsupervised learning**
A machine learning method where no teacher is involved e.g. the self-organization is a sort of unsupervised learning.

**vector quantization**
A unifying name for methods which quantize some input dataset through a number of reference or code-book vectors and also for the associated winner search mechanism (such as the distance based nearest neighbor search).

**weight space**
A vector space which contains vectors created by all weight of all neurons. Usually, it is the space above which the error function is defined.

**winner-takes-all / winner-takes-most**
Adaptation strategies of competitive networks. They differs in influence to the neighborhood of the excited unit, i.e. if the adaptation is taken by the winner only or if a part of adaptation is forwarded also to its neighbors.
**List of abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute Unified Device Architecture</td>
</tr>
<tr>
<td>FBA</td>
<td>Force-Based Algorithm</td>
</tr>
<tr>
<td>GNG</td>
<td>Growing Neural Gas</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>JGNG</td>
<td>Joined Growing Neural Gas</td>
</tr>
<tr>
<td>LVQ</td>
<td>Learning Vector Quantization</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi Layer Perceptron</td>
</tr>
<tr>
<td>NG</td>
<td>Neural Gas</td>
</tr>
<tr>
<td>NN</td>
<td>Nearest Neighbor</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
</tr>
<tr>
<td>SGNG</td>
<td>Supervised Growing Neural Gas</td>
</tr>
<tr>
<td>SOM</td>
<td>Self-Organizing Feature Map</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
</tbody>
</table>
8.3 Activities and publications

This section contains information about my recent activities and projects I have collaborated on. A Summary of my publication work is also enlisted.

**Media Research Lab**
- Advisor: doc. Dr. Ing. Eduard Sojka
- Collaboration on project *Anonymisation of Images Captured By In-Car and Stationary Cameras* for Volkswagen AG

**Floreon**
- Collaboration on project *Floreon - The System for Emergent Flood Prediction* for Moravia-Silesian Regional Office.
- Analysis and implementation of high-scale spatial indexing tool for traffic data analysis built on unique raster indexing method.
- Analysis and implementation of application for visualization of terrain & floods. The research was focused on the capability of processing high amount of spatial data (up to TB) and real time visualization of such data.

**Research Laboratory of Intelligent Systems**
- Advisor: doc. RNDr. Marie Duží, CSc.
- Collaboration on research project No. 1ET101940420 *Logic and Artificial Intelligence for Multi-agent Systems* supported by the program *Information Society* of the Czech Academy of Sciences.
- Research focused on spatial orientation capability of autonomous agents based on dynamically generated behavioral patterns.

**ITA Technologies & Software**
- Optimization of an existing model for temperature control of hot strip rolling process.
- Analysis of measured data and identification of possible targets for optimization.
- Design and ongoing implementation of global scope optimization method driven by evolitional approach.

**Amfora Research Group**
- Advisor: prof. RNDr. Václav Snášel, CSc.
- Collaboration on research project *Flexible Neural Trees*.
- Development of tools for visualization of neural networks and evolitional algorithms.
Publications


