Finding Optimal Parameters for Neural Gas Networks Using Evolutionary Algorithms

Schriftliche Prüfungsarbeit
für die Bachelor-Prüfung des Studiengangs
Angewandte Informatik
an der
Ruhr-Universität Bochum

vorgelegt von

Lomp, Oliver
Matrikelnummer: 108005204827

Abgabedatum: 21. Oktober 2008

Erstprüfer: Rolf Würtz
Zweitprüfer: Christian Igel
Betreuer: Guillermo S. Donatti

Contents

Abstract

The parameter values used for the Growing Neural Gas (GNG) algorithm are generally determined empirically. This requires long calculation times and may lead to values which are not optimized for the data set they are being used with. The present work proposes the use of Evolutionary Algorithms to optimize these parameter values. During the optimization process, GNG networks are created with the parameter values stored in individuals from the Evolutionary Algorithm, and trained with object features extracted from images of the ETH-80 database. An individual's fitness is calculated according to three different functions which assess the performance of the trained GNG networks. A feature-based object recognition and categorization model defined by a taxonomic hierarchy of self-organized GNG networks is trained and tested using the parameter values obtained from the optimizer, and with the same data sets employed during the optimization process. The categorization and recognition rates of the obtained models are compared to the ones achieved with the empirically set parameter values. The results suggest that the optimization of parameter values significantly increases the model's performance when using the constrained initial values for the individuals in the first parent population of the Evolutionary Algorithm. Using unconstrained initial values seems to lead to parameter values that, when used in the GNG algorithm, create unstable networks, which are incapable of learning. This is likely caused by the existence of local optima throughout the parameter search space.

1 Introduction

Machine learning algorithms often have a high number of parameters that determine their performance. The Growing Neural Gas for example, is an algorithm capable of adapting the nodes of an undirected graph to the topology of an arbitrary probability distribution. However, depending on the setup of its parameters, it could contain too many nodes, or too few, thus not correctly learning the topology of the input distribution.

Currently, the values of these parameters are set empirically, often by applying prior knowledge about the nature of the data to be learned. Not only is finding the parameter values for a specific data set time consuming, but it is also possible that the values found are unoptimized. Therefore, the present work proposes the use of Evolutionary Algorithms for performing this task. These algorithms are based on the Darwinian principles of evolution and evolve possible solutions for a problem by encoding them into a genome,

which, over many generations, is mutated and recombined to form an optimal solution.

The present work focuses on optimizing the parameter values of the Growing Neural Gas algorithm trained with features extracted from images of the ETH-80 database [Leibe and Schiele, 2003]. This database is also used to train and test a feature-based object recognition and categorization model introduced by Guillermo S. Donatti (manuscript in preparation, 2008); this model is defined by a taxonomic hierarchy of Growing Neural Gas networks, and therefore can be used to evaluate the effect of the optimized parameter values on the model's performance.

In the Evolutionary Algorithm, Growing Neural Gas networks are trained using the parameter values provided by the Evolutionary Algorithm, and their performance is evaluated with three different fitness functions. Based on these fitness values, the Evolutionary Algorithm selects individuals (i.e., the parameter value combinations) that survive onto the next generation. Hence, these fitness functions greatly influence the quality of the parameter values (i.e., the capability of networks created with them to learn the distribution of the input samples) resulting from the optimization process.

Results obtained using optimized parameter values in the feature-based object recognition and categorization model suggest that the method proposed in the present work is capable of improving parameter values, as long as the initial values used for the optimization process are restricted. In this case, the model's object recognition and categorization rates can be improved significantly.

2 Neural Gas Algorithms

2.1 Introduction

Neural Gas [Martinetz and Schulten, 1991] is an algorithm that adapts the structure of an undirected graph to the topology of an arbitrarily structured input manifold. During this process, input samples are generated from this manifold and presented to the algorithm. For each input sample, a configurable number of the network's closest nodes is adapted by moving them towards it. In the initial iterations, many nodes are moved towards the input sample. With each input sample, the amount of nodes influenced by it decreases. During the execution of the algorithm the number of nodes in the Neural Gas network stays constant, while the edges are adapted according to a learning rule similar to Competitive Hebbian Learning: each time an input sample is presented, a new connection is created between its two closest

nodes.

The Growing Neural Gas algorithm [Fritzke, 1995] extends the Neural Gas algorithm by adding a local error measure for each node. This error is accumulated based on the node's distance to the input samples presented to the network. Regularly, a new node is inserted between the two nodes that have accumulated the largest amount of error. The underlying graph of the Growing Neural Gas network starts with two nodes, and expands until it reaches a predefined maximum number of nodes. Therefore, the number of nodes in the Growing Neural Gas network is no longer fixed as it is in the Neural Gas algorithm.

A second addition proposed by [Fritzke, 1997] is the utility measure. It is used to determine the usefulness of each node by estimating the increase of the global network error if the node would not be present. Based on their utility value, the nodes that contribute little to reducing the global network error can be removed.

The Growing Neural Gas algorithm is an unsupervised learning algorithm, because it only takes an input sample, but no associated target output. [Bolder, 2005] proposes a method to extend the Growing Neural Gas to a supervised algorithm capable of solving classification problems: each node stores a matrix, the local linear mapping, used for mapping from the input space to the output space. Inputs for this version consist of an input sample and a target output. This mapping can also be reversed by storing a second matrix in the nodes, leading to a bidirectional mapping.

[Bolder, 2005] also introduces Bootstrapping, a method that inserts nodes at the positions of the input samples of a subset of the training samples presented to the network. This method can significantly increase the learning speed of the Growing Neural Gas algorithm, because the topological information from the input samples of the subset is preserved. It would otherwise be lost, because the initial two nodes of the network would only move in the general direction of the input samples.

2.2 Neural Gas

[Martinetz and Schulten, 1991] propose the *Neural Gas* network, an unsupervised neural network based on the self organizing feature map model introduced by [Kohonen, 1982]. Neural Gas is a method for adapting the structure of an undirected graph to the topology of an arbitrarily structured manifold. It minimizes the vector quantization[1] error of a set of reference

[1] *Vector quantization* is the process of finding a set of reference vectors $R = \{x_i \in \mathbb{R}^{d_I}\}$ (where d_I is the dimension of the input space) that represent a potentially infinite data

vectors $X = \{x_i | i \in \{0 \cdots N\}\}$ extracted from the manifold M. These reference vectors are associated with the nodes of the network, and when fully trained, they preserve the topology of the input manifold.

For every input sample generated from M, the structure of the Neural Gas network is adapted by moving a number of the nodes closest to it (i.e., with the lowest distance values) towards the input sample. In initial iterations, a large number of nodes is moved. In subsequent iterations this number decreases until only a few nodes are adapted.

The amount by which the nodes are moved depends on their distance to the input sample. This is calculated using a distance measure d, and by applying a learning rule similar to Hebbian Learning combined with a time-decay of the centers x_i (i.e., the reference vectors) of the nodes from the previous iteration:

$$\Delta x_i = \epsilon(t) \cdot f_i(D_v) \cdot (v - x_i) \tag{2.1}$$

where $\epsilon(t) \in [0, 1]$ is a step width decreasing with each input sample presented, and $f_i(D_v)$ is a function calculating the excitation of the node i with the set of distances D_v induced by the input sample v.

The excitation of each node is based on its rank in the list of nodes sorted according to their distance from the current input sample. Each node is assigned a rank i_k, meaning there are k nodes closer to the input sample. Thus, i_0 is the node closest to the input, hereafter also referred to as the best-matching or winning node. Similarly, i_1 is also referred to as the second-best node.

In addition to adapting the nodes of the network, the algorithm develops connections (i.e., edges in the graph) between the nodes. These edges are stored in a *connection matrix* C_{ij}, which asymptotically approximates the neighborhood relationships of the Voronoi-Polygons[2] described by the nodes. Similar to Competitive Hebbian Learning[3], for each input presented, the best matching and second best matching nodes are connected by adding a new edge between them. These edges age over time and are removed if they

set in a submanifold $M_{sub} \subseteq M$ (with M being a manifold). These reference vectors have to be selected to compress the dataset but allow for its reconstruction with a minimum amount of error. [Martinetz and Schulten, 1991]

[2] A Voronoi Polygon P_r describes the region of a space in which all vectors lie closer to a reference point r than to any other in a given set R of reference vectors. [Martinetz and Schulten, 1991]

[3] *Competitive Hebbian Learning* is a learning rule that finds connections between a set of centers C in $\mathbb{R}^n$ that represent the topology of a data distribution $P(\xi)$ in the input space by inserting a connection between the centers closest and second closest to the current input sample[Fritzke, 1995].

exceed a maximum age. For a large number of input samples the algorithm described above results in a minimization of the vector quantization error and the modification of the topology of the network to approximate a subgraph of the Delaunay triangulation[4] called the *induced Delaunay triangulation*[5]. According to [Fritzke, 1995], this subgraph optimally preserves the topology of the input manifold for all areas in the input space where $P(\xi) > 0$ (with $P(\xi)$ being the probability distribution of the input samples).

Like any vector quantization method, Neural Gas can also serve to determine a clustering of the distribution describing the manifold. Therefore, another important application of the Neural Gas algorithm is classification. In this context, the nodes represent the different classes, and the class for an input sample is found by determining its closest node.

2.2.1 The Neural Gas Algorithm

The formalized version of the Neural Gas algorithm is described as follows:

1. Initially, the centers x_i of all nodes are set to random values and all connection ages T_{ij} as well as all connections C_{ij} are set to zero:

$$T_{ij} = 0,\ C_{ij} = 0\ \forall i, j \in \{0, \ldots, N\} \tag{2.2}$$

   where N is the number of nodes in the network.

2. An input vector v is generated from the submanifold $M_{sub} \subseteq M$.

3. The nodes are sorted according to the distance measure by generating the sequence of indices $ranking = (i_0, i_1, \ldots, i_{N-1})$ with

$$d(v, x_{i_0}) < d(v, x_{i_1}) < \ldots < d(v, x_{i_{N-1}}) \tag{2.3}$$

   where d is the distance measure.

4. The centers of all nodes are adapted by moving them towards the input signal. How much they are moved depends on their previously calculated ranking index:

$$\Delta x_i = \epsilon \cdot e^{\frac{-k_i}{\alpha}} \cdot (v - x_i^{(old)}) \tag{2.4}$$

[4]The Delaunay triangulation of a set of points is the graph where only those points are connected which have neighboring Voronoi polygons [Fritzke, 1995].

[5]The induced Delaunay triangulation is a subgraph of the Delaunay Triangulation limited to the regions where the distribution of the input samples has a value greater than zero[Fritzke, 1995].

where k_i is the ranking index of the node i in the *ranking* (e.g., the node at i_0 has the rank 0); thus, the nodes close to the input are affected more than the ones that are further away. $\epsilon \cdot e^{\frac{-k_i}{\alpha}}$ is the realization of the function $f_i(D_v)$ from equation 2.1, the excitation of the node i for a given set of distances. It decreases proportional to the rank of each node. By using an exponential function, only the nodes with the highest ranks receive an excitation significant enough to be moved. The value of α decays over time, leading to a strong excitation for all nodes when the first input samples are presented but only for nodes close to input samples presented later.

5. A new connection is created (or, if one already exists, its age is set to zero) between the nodes i_0 and i_1:

$$C_{i_0 i_1} := 1, T_{i_0 i_1} := 0 \tag{2.5}$$

6. The connection ages are increased. To improve performance, only the ages of connections adjacent to i_0 are increased:

$$T'_{i_0 j} = T_{i_0 j} + 1 \ \forall j \in \{j | C_{i_0 j} = 1\} \tag{2.6}$$

   This reduces the necessary amount of computation, and is equivalent to incrementing all ages when a large number of training samples is presented, as long as all nodes have the same probability of being the winner node [Martinetz and Schulten, 1991].

7. All edges which have an age greater than T_{max} are removed

$$C_{ij} = 1 \wedge T_{ij} > T_{max}(t) \rightarrow C_{ij} := 0 \tag{2.7}$$

8. The algorithm is repeated from step 2, until a convergence criterion (e.g., a maximum number of input samples presented to the network) is met.

2.3 Growing Neural Gas

The Neural Gas algorithm uses parameters depending on the number of input samples presented to the network, namely ϵ, α and T_{max}. This makes continuous learning difficult, because after a certain number of iterations the network is relatively fixed. Additionally, the number of nodes is the same for the entire runtime of the algorithm. To eliminate these limitations, [Fritzke, 1995] proposes a modified version of the algorithm, the Growing Neural Gas. This version uses parameters that are constant over time, and it is capable of dynamically adding and removing nodes.

2.3.1 The Local Error Measure

Each node i in the Growing Neural Gas network stores an additional value e_i containing a local error measure used to determine which nodes generate high errors. The error of the node i_0 closest to the current input sample v is updated according to

$$\Delta e_{i_0} := ||x_{i_0} - v||^2 \tag{2.8}$$

where x_{i_0} is the current center in the input space of the node i_0. For each input sample, the error values of all nodes are decreased by a factor α_{error}.

Regularly, the node q with the maximum accumulated error is determined, as well as the node s with the highest accumulated error among the neighbors of q (i.e., the nodes connected to q with an edge). Once they are identified, a new node r is inserted between q and s. The new center of r lies between these two nodes:

$$x_r := \frac{1}{2}(x_q + x_s) \tag{2.9}$$

The edge between q and s is removed, and edges connecting q and r as well as r and s are inserted, placing r between the two nodes in the topology of the network:

$$C_{qr} = 1 \tag{2.10}$$

$$C_{rs} = 1 \tag{2.11}$$

Additionally, the error values of q and s are decreased by a factor α_{growth}:

$$e_q = \alpha_{growth} \cdot e_q^{(old)} \tag{2.12}$$

$$e_s = \alpha_{growth} \cdot e_s^{(old)} \tag{2.13}$$

Therefore, the new node is created in a region in which the current nodes often lie far from the input samples, thus accumulating a high local error value. The new node reduces the global error by reducing the sizes of the Voronoi polygons of the nodes that are already in the network.

2.3.2 Isolated Nodes

Removing an edge (e.g., because it has an age greater than the maximum age) potentially leaves nodes isolated in the network. In this case, the isolated nodes are removed as well, as long as the number of nodes in the network does not fall below the minimum (i.e., two) required for the algorithm.

2.4 Utility measure

For stationary or slowly changing distributions, the nodes in a network trained with the Growing Neural Gas algorithm approximate an optimal set of reference vectors of the input manifold. However, if the distribution of the input samples changes rapidly over time (this often occurs in natural processes), the aforementioned network often retains nodes that no longer lie in areas where $P(\xi) > 0$ (i.e., areas from which input samples are no longer generated). These nodes, called "dead units"[Fritzke, 1997], no longer have a chance to become the winner or second best node. Since only the connections of the nodes close to the input samples are updated, the dead units will no longer be updated and are "stranded". These nodes no longer contribute to the network and take up network resources (i.e., the possibility of creating a new node in a region of high error).

To make the Growing Neural Gas adapt to non stationary distributions, [Fritzke, 1997] introduces a utility measure u_i for each node in the network. The purpose of this measure is to estimate how much the global network error would increase if the node i would be removed. This error difference is accumulated in each iteration (i.e., for each input sample v presented to the network). When the best matching node i_0 and the second best matching node i_1 have been determined, the utility measure of the node i_0 is updated:

$$\Delta u_{i_0} := ||v - x_{i_1}||^2 - ||v - x_{i_0}||^2. \tag{2.14}$$

Recall, that the local error measure for each node is calculated according to $\Delta e_i := ||v - x_i||^2$. Thus, the utility in each iteration is increased[6] by the difference between the errors of the winner and the second best node. Additionally, the utility values of all nodes are decayed by a factor $\alpha_{utility}$ after each iteration. Therefore, nodes have a small utility when one of the following two conditions is fullfilled: The first one is when a node is close[7] to another one (i.e., the winner node and the second best node are relatively close in the input space). In this case, removing the winner does not result in a significant change of the global network error, because the second best node takes over all the input samples that would otherwise fall into the Voronoi region of the node that is removed. Furthermore, Δu_{i_1} is always small, since Δe_{i_0} and Δe_{i_1} have almost the same value (i.e., the difference between them is small, see equation 2.14). In the second case, the node lies in one of the regions of the probability distribution where it no longer, or only rarely,

[6]The change of the utility will always be positive because $e_{i_0} < e_{i_1}$, since per definition the winner is always closest to the input.

[7]Close in the sense that the distance measure has a small value compared to the distance between other nodes.

becomes the winner. Thus, its utility no longer increases, and due to the decay after each generation it becomes smaller over time.

In addition to calculating the utility value, a criterion is needed to determine which nodes can be removed based on it. Removing the node with the lowest utility value is not sufficient, because that results in the removal of a node, even if all nodes are actually useful to the network.

[Fritzke, 1997] proposes to remove a node i, when the following holds true: i is the node with the minimal utility value, i.e.

$$i := \underset{j}{argmin} \{u_j \mid j \in \{0, \ldots, N\}\} \tag{2.15}$$

and

$$\frac{e_{max}}{u_i} > \kappa, \tag{2.16}$$

where e_{max} is the maximal accumulated error, u_i the utility of the node i, and κ has any positive value.

This criterion is based on the fact that removing a useless node allows the network to create a new one in the region of the highest error. Since the utility of the useless node approximates how much the network error increases if it is removed, and new nodes are always inserted where the error is highest, $\frac{e_{max}}{u_i}$ approximates the amount by which the error would be reduced when removing the useless node.

2.5 Supervised Growing Neural Gas

2.5.1 The Local Linear Mapping

A *local linear mapping* is employed to introduce a supervised variant of the Growing Neural Gas algorithm[Bolder, 2005], allowing nodes to learn a mapping from the input space to the output space. Each node i stores a matrix Y_i used to calculate its mapping, and a center y_i in the output space.

The output of the node for an input sample v can then be calculated as follows:

$$o_i(v) := y_i + Y_i \cdot (v - x_i) \tag{2.17}$$

Where x_i is the center of the node i in the input space.

As is common for supervised learning, the samples provided during the learning phase consist of two elements: the input sample v, and the target output v'. For each input sample, the best matching node i_0 is determined as follows:

$$i_0 := \underset{i}{argmin}\{d(x_i, v) \mid i \in \{0, \cdots, N\}\} \tag{2.18}$$

The matrix and the center in the output space are moved towards the target output by the step width ϵ_{output}:

$$\Delta Y_{i_0} := \epsilon_{output} \cdot (v' - o_{i_0}(v)) \cdot (v - x_{i_0})^+ \quad (2.19)$$

$$\Delta y_{i_0} := \epsilon_{output} \cdot (v' - o_{i_0}(v)) \quad (2.20)$$

2.5.2 Bidirectional Mapping

The mapping described in section 2.5.1 is a unidirectional local linear mapping from the input space to the output space. Reversing it (i.e., making it a *bidirectional mapping*) could be achieved by creating a second Growing Neural Gas network where input and output change places. However, integrating the reverse mapping into the Growing Neural Gas algorithm proves to be more effective in certain cases, and never worse [Bolder, 2005].

To integrate the mapping, every node i in the network is assigned a new matrix X_i. This matrix is used for calculating the reverse mapping for any point $\bar{v}$ in the output space according to the following equation:

$$\bar{o}(\bar{v}) := x_{i_0} + X_{i_0} \cdot (\bar{v} - y_{i_0}), \quad (2.21)$$

where i_0 is the winner node (i.e., the node closest to $\bar{v}$ in the output space).

Accordingly, the matrix X_{i_0} is calculated every time an input (v, v') is presented:

$$\Delta X_{i_0} := \epsilon_{winner} \cdot (v - \bar{o}(v')) \cdot (v' - y_{i_0}). \quad (2.22)$$

where ϵ_{winner} is a predetermined step width.

2.6 Bootstrapping

With the aim to increase the learning speed in initial iterations, [Bolder, 2005] proposes the usage of a *bootstrapping* method. The first N_{boot} input samples are used to set the components of new nodes: for each training sample v_t with the target output v'_t (where $t < N_{boot}$), a new node t is created with the following components:

$$x_t = v_t \quad (2.23)$$

$$y_t = v'_t \quad (2.24)$$

where x_t is the center in the input space, and y_t is the center in the output space of the node t. The matrix Y_t is calculated analogously to equation 2.22.

Since without bootstrapping the network begins with two nodes, data is lost until the network creates sufficient nodes to represent the distribution of

the input samples. The amount of information lost in the initial iterations of the network increases with the number of dimensions of the input and output spaces. Therefore, the effectiveness of bootstrapping is directly proportional to the dimension of the data distribution.

2.7 The Growing Neural Gas Algorithm

The Growing Neural Gas algorithm presented here takes into account all additions presented in sections 2.3.1–2.6. However, only the unsupervised version of this algorithm with the utility measure and the local error measure, but without bootstrapping is used during the optimization process described in section 4.

1. The network is initialized with two connected nodes. Their components are set to random values; depending on the version of the algorithm that is used, each node i contains different values: x_i, the center in the input space, e_i the local error value, and the utility value u_i. When using a local linear mapping (i.e., using the network as a classifier), the nodes also include a matrix Y_i; if this mapping is bidirectional, an additional matrix X_i is stored as well.

2. A new training input sample is determined. It consists of a vector v generated from the distribution of the input submanifold $M_{sub} \subseteq M$. During the learning phase of the supervised algorithm, the input sample also contains a target output v'.

3. If bootstrapping is used, then the first N_{boot} training input samples are used to insert a new node with the following component values:

$$x_t := v \tag{2.25}$$
$$e_t := 0 \tag{2.26}$$
$$u_t := 0 \tag{2.27}$$

   and additionally, for the supervised algorithm:

$$y_t := v' \tag{2.28}$$

   where t is the number of the current iteration of the algorithm.

   As long as the input samples are used for bootstrapping (i.e., as long as the number of network iterations $t < N_{boot}$), the algorithm continues from step 2, otherwise it is continued from step 4.

4. The winner i_0 and second best node i_1 for the input sample v are determined using the distance measure d as follows:

$$i_0 = \underset{i}{argmin}\, d(x_i, v) \tag{2.29}$$

$$i_1 = \underset{i \neq i_0}{argmin}\, d(x_i, v) \tag{2.30}$$

   If a reverse mapping is included, then instead of the distance measure d_i, the measure described in the following equation is used:

$$\bar{d}_i(v') = ||v' - y_i||^2 \tag{2.31}$$

   During the learning stage (i.e., while target outputs are still known), the measure needs to take into account the target output:

$$\bar{d}'(v, v') = ||v - x_i||^2 + ||v' - y_i||^2 \tag{2.32}$$

5. The ages $T_{i_0 j}$ of all edges neighboring[8] the node i_0 are incremented.

6. The age of the edge between i_0 and i_1 is set to zero. If no edge exists between the two, a new one is created.

7. The error measure e_{i_0} of the winner node is updated. For the unsupervised variant, the squared network error is added:

$$\Delta e_{i_0} = ||v - x_{i_0}||^2 \tag{2.33}$$

   For the supervised variant, the squared distance to the network output is added instead:

$$\Delta e_{i_0} = ||v' - o_{i_0}(v)||^2 \tag{2.34}$$

   For a bidirectional mapping, the equation 2.34 is extended to:

$$\Delta e_{i_0} = ||v' - o_{i_0}(v)||^2 + ||v - \bar{o}_{i_0}(v')||^2 \tag{2.35}$$

8. The utility of the winner node is updated. For the unsupervised version, the increase of the global error that occurs if the winner node would not be present is added:

$$\Delta u_{i_0} = ||v - x_{i_1}||^2 - ||v - x_{i_0}||^2 \tag{2.36}$$

   For the supervised algorithm, the utility is updated using the center in the output space instead:

$$\Delta u_{i_0} = ||v' - o_{i_1}(v)||^2 - ||v' - o_{i_0}(v)||^2 \tag{2.37}$$

[8]Here, the neighborhood of a node are those nodes connected to it through an edge.

9. The network's topology is adapted. The winner node is moved towards the input sample, and its direct neighbors i_n are adapted according to

$$\Delta x_{i_0} = \epsilon_{winner} \cdot (v - x_{i_0}) \tag{2.38}$$
$$\Delta x_{i_n} = \epsilon_{neighbor} \cdot (v - x_{i_n}) \tag{2.39}$$

where ϵ_{winner} and $\epsilon_{neighbor}$ are predetermined step widths.

For the supervised version, the center in the output and the matrix describing the local linear mapping need to be changed as well:

$$\Delta y_{i_0} = \epsilon_{output} \cdot (v' - y_{i_0}) \tag{2.40}$$
$$\Delta Y_{i_0} = \epsilon_{output} \cdot (v' - o_{i_0}(v)) \cdot (v - x_{i_0})^+ \tag{2.41}$$

where ϵ_{output} is a step width.

For the reverse mapping, the matrix X_{i_0} and the neighbors y_n in the output space are adapted using the following equations:

$$\Delta X_{i_0} = \epsilon_{winner} \cdot (v - \bar{o}_{i_0}(v')) \cdot (v' - y_{i_0})^+ \tag{2.42}$$
$$\Delta y_n = \epsilon_{neighbor} \cdot (v' - y_n), \tag{2.43}$$

and the step widths ϵ_{winner} and ϵ_{output} have the same value.

10. All edges that have an age greater than the maximum age are removed, and all resulting isolated nodes are removed as well, as long as the network has at least two nodes.

11. For every λ_{growth} input samples presented to the network, a new node is inserted. For this, the node q with the highest error, and the node r with the highest error among its neighbors are determined. A new node g with

$$x_g := \frac{1}{2}(x_q + x_r) \tag{2.44}$$
$$e_g := e_q \tag{2.45}$$
$$u_g := u_q \tag{2.46}$$

is inserted. For a supervised network, the center in the output space and the matrix containing the mapping are set to

$$y_g := \frac{1}{2}(y_q + y_r) \tag{2.47}$$
$$Y_g := \frac{1}{2}(Y_q + Y_r) \tag{2.48}$$

and for bidirectional mapping the matrix X_g is calculated accordingly:

$$X_g := \frac{1}{2}(X_q + X_r) \tag{2.49}$$

After inserting the node, the edge between r and q is removed, and edges from g to r and q are created. Afterwards the error values of the old nodes are decayed by α_{growth}.

12. For every λ_{decay} input samples presented to the network, a node is deleted based on its utility value.

    For this, the maximal error value e_{max} is determined. The node $i_{u_{min}}$ with the lowest utility value is deleted if its utility is smaller than a fraction of e_{max}, i.e., if the equation

    $$u_{i_{u_{min}}} \cdot \kappa < e_{max} \tag{2.50}$$

    holds true.

13. All error and utility values are decayed by multiplying them with a factor $\alpha_{error,utility} \in (0, 1]$.

14. The algorithm is repeated from step 2 onwards, until a convergence criterion is met, or all training samples have been presented. However it is also possible to let the network learn indefinitely.

3 Evolutionary Algorithms

3.1 Introduction

Evolutionary Algorithms are a class of stochastic search algorithms based on the principles of Darwinian evolution. These algorithms are capable of optimizing problems for which a fitness value can be defined. During the optimization, different generations of individuals, composed of genomes containing possible solutions, compete with each other for survival. In every generation, each individual is assigned a fitness value, which indicates its suitability to solve the problem. Based on this value and stochastic influences, some individuals are selected to survive onto the next generation and to generate an offspring by the recombination of their genomes. This process is repeated until a satisfying solution has been found (e.g., an individual with a fitness value lower than a preset threshold).

Evolution Strategies are a subclass of Evolutionary Algorithms. The Covariance Matrix Adaptation strategy uses a normal distribution with an adapted covariance matrix for mutating the offspring individuals based on the genomes of a single parent. The distribution's covariance matrix is adapted to make the mutations more likely to occur in the same direction as the mutations that increased the fitness values of individuals in previous generations.

3.2 Populations and Individuals

Evolutionary Algorithms optimize a problem by evaluating some of its possible solutions against each other. Each of these is encoded into the chromosomes of an individual, and the resulting set of individuals forms a population.

The premise for using Evolutionary Algorithms is that a fitness value can be assigned to every individual (i.e. a fitness value $f(x)$ can be assigned to every possible solution x). This fitness value serves as a basis for selecting the individuals that survive onto the next generation. For example, let w be a vector of weights representing the connections between neurons of a neural net set to be optimized by an Evolutionary Algorithm. If the goal of the network using w is to represent a function g, the fitness value could be a measure of the network error for a data set (e.g., the squared network error for samples from g in a predetermined range). During the optimization the Evolutionary Algorithm would create various individuals, and would calculate the network error for each one to acquire its fitness value.

There are several approaches to encoding a possible solution in a chromosome (e.g., character or bitstrings). Selecting the encoding approach depends on the type of data and the Evolutionary Algorithm to be used. In the present work, the chromosomes are considered an array of integral or real values, because the Neural Gas algorithm contains only real and integral valued parameters.

3.3 The Evolutionary Cycle

In essence, Evolutionary Algorithms always follow the same structure [Kreutz et al., 2008]. They hold a population of individuals, which is *initialized* (e.g., by assigning random values to the chromosomes of its individuals); after that, the individuals are *evaluated* (i.e., each individual is assigned a fitness value based on its genome). Subsequently, the population for the next generation is *selected* from the individuals of the last one. Then, *mating selection and recombination* are performed to form additional individuals.

Lastly, these additional individuals are *evaluated*, and the cycle is repeated until a termination criterion (e.g., a predetermined fitness value) is met.

The algorithm described above is a basic form of Evolutionary Algorithms, containing elements common to most variations. However, mutation is an additional step used for the present work applied when the offspring individuals have been generated. In general, it is part of the recombination process which allows to explore the search space by adding a random element to the formation of the individuals' genomes.

3.3.1 (μ, λ) and $(\mu + \lambda)$ Selection

Selection occurs twice in the evolutionary cycle: the mating selection and the selection of the individuals for the next generation. The latter one is the process of determining which individuals survive and are passed on to the next generation, and its mechanisms are the focus of this section.

The selection mechanisms presented in this section are deterministic. There are also stochastic selection mechanisms which assign individuals a probability to survive that is directly proportional to their fitness to solve the problem that is being optimized. The present work only uses deterministic selection mechanisms.

Several different kinds of deterministic selection mechanisms are proposed in the literature. In the present work, only (μ, λ), $(\mu+\lambda)$ and elitist selection are used. In these selection mechanisms, there are two kinds of populations during each generation. The first one is the parent population, which consists of μ individuals from the previous generations. The second one is the offspring population containing λ individuals, which are created by mating the individuals from the parent population followed by mutating the resulting offspring.

Each individual survives based on its fitness value. When applying (μ, λ) selection, only the μ fittest individuals from the offspring generation become part of the next parent population. This selection mechanism does not preserve the best solutions over generations, because the offspring individual are not necessarily fitter than their parents. However, such a property allows this mechanism to have a greater chance of finding the global optimum, because it is capable of moving away from a locally optimal solution. In the case of $(\mu + \lambda)$ selection, the next generation consists of individuals from both the offspring and the parent populations of the preceding generation. Opposed to the (μ, λ) selection mechanism, this method always preserves the best solutions found. If none of the offspring individuals are fitter than their parents, in the next generation the parent population remains the same. This difference to the (μ, λ) selection mechanism reduces the chances of finding

a global optimum, but also decreases the chances of losing a good solution once it is found.

Therefore, both methods have an advantage over the other. The (μ, λ) selection mechanism performs exploration while the $(\mu + \lambda)$ one preserves the best solutions found. To make use of both properties, the present work uses a selection mechanism that combines the two[Kreutz et al., 2008] mechanisms. In principle, this selection mechanism selects individuals similar to the (μ, λ) mechanism. However, prior to selecting the best offspring individuals, $n_{elitists}$ from both the parent and the offspring population are selected to survive. After those individuals are selected, $\mu - n_{elitists}$ of the fittest individuals from the offspring population are selected. It is relevant to note, that no individual gets selected twice. This combined selection mechanism preserves good solutions for the problem, even if all the offspring individuals have a lower fitness value than their parents. Additionally, it enables exploration, because a number of the best offspring individuals still survives, giving the offspring population the chance to evolve towards a different local optimum.

3.3.2 Mutation and recombination

Recombination is the process of combining individuals from the parent population to form new offspring individuals. During this process, the parents can be selected in several ways. In the present work, each individual in the parent population has the same chance of being one of the two parents of a new individual.

After the parents are selected, they are combined to form a new individual by performing a *crossover* between their genomes. The result of the crossover is a new genome containing parts from both parents: the genomes of both parents have the same length (i.e., they store the same number of values). The crossover algorithm starts with the values from one of the parent individuals and writes them into the new genome. After a number of values, it switches to the second parent and uses the values from its genome. The number of cross points $n_{crosses}$ is determined (e.g., randomly) and the process described above is repeated $n_{crosses}$ times, resulting in a genome that is a combination of both parents.

After the offspring individuals are created, each of their genomes is mutated by drawing a vector from a normal distribution centered around zero and adding it to the genome of the individual. The variance of the normal distribution can be varied to achieve more or less exploration of the search space during the evolutionary cycle (i.e., a greater value for the variance results in more exploration while a smaller one restricts the mutation of the individuals to a smaller range).

3.4 Covariance Matrix Adaptation

The Covariance Matrix Adaptation (CMA) is an Evolution Strategy devised for optimizing problems with solutions consisting entirely of real values. CMA uses a modified normal distribution for mutating the genomes: the distribution's covariance matrix is adapted according to the information about the search space obtained over the course of the last generations. It uses one parent from which all offspring individuals are generated by means of mutation[Igel et al., 2006]: without loss of generality, assume $c_p^g \in \mathbb{R}^d$ to be the representation of the parent's chromosome. The offspring i is generated according to the following equation:

$$c_{o_i} := c_p^g + m_i \tag{3.1}$$

where m represents the value of the mutation. Using a n-dimensional normally distributed random variable with a unit covariance matrix for generating m would result in the method described in section 3.3.2. However, For the Covariance Matrix Adaptation the mutation is sampled with a covariance matrix C^{g+1}:

$$m \sim N(0, C^{g+1}) \tag{3.2}$$

C^{g+1} is initially set to 1. For each subsequent generation it is updated orienting the random distribution towards solutions that are proven to have better fitness values. For example, if a parent solution s_1 was mutated to become an offspring s_2, and $f(s_2) > f(s_1)$ (i.e., s_2 would become the new parent in the next generation), then the covariance matrix is adapted making a sample more likely to occur in the direction of $s_2 - s_1$ (and in the opposite direction as well, since the normal distribution is symmetric). In principle, this method is similar to a gradient descent, with the difference that the movement towards an optimal solution is not deterministic.

Due to its nature, the CMA Evolution Strategy can find an optimum in significantly less generations than Evolutionary Algorithms when dealing with real valued parameters. It can also be used to optimize multiple goals of one problem [Igel et al., 2007], however this could not be used in the present work.

4 Optimizing Parameter Sets

4.1 Introduction

Many machine learning algorithms, such as the one introduced in section 2.1, have a high number of parameters that control the learning process.

The values of these parameters need to be selected carefully to achieve the desired learning effects, either by applying prior knowledge about the nature of the problem presented to the algorithm or by setting them empirically. In most cases, a combination of these methods is used, and the search for optimal values for these parameters becomes extensive and time consuming.

Evolutionary Algorithms on the other hand are capable of optimizing any problem for which a fitness function can be defined.

Since the performance of machine learning algorithms can be evaluated (e.g., by using the squared network error of a multilayer neural network), the present work proposes a method for using Evolutionary Algorithms to optimize their parameter values.

4.2 Optimizing Growing Neural Gas Networks

The present work focuses on optimizing the parameter values of the Growing Neural Gas algorithm because of the high number of parameters used by this algorithm, which makes finding optimal values a very complex task. Additionally, applying it to represent the visual knowledge of a feature-based object recognition and categorization model offers a chance to evaluate the practical effect of the parameter values found during the optimization process on the object recognition and categorization rates. Since the fitness values obtained from the Evolutionary Algorithm with different fitness functions presented in section 4.5 cannot be compared, the object recognition and recall rates can be used for this as well.

4.3 The Optimizer

The parameter optimization is achieved with the evolutionary cycle described in section 3.3. Special considerations have to be taken into account in order to evaluate parameter values of Growing Neural Gas networks. Among those is the definition of a fitness function and the implementation of the range restrictions of the parameters to be optimized. These considerations are addressed in the following algorithm used for optimizing the parameter values of the Growing Neural Gas algorithm:

1. The population sizes are selected, μ for the parent population and λ for the offspring population, as well as the number of elitists in the parent population and a standard deviation σ for the normal distribution used when the individuals are mutated. The parameters which have real numbers ranging from zero to one (e.g., the error decay α_{error}, see step 13 in section 2.7), use a different deviation $\sigma_{[0,1]}$, because the magnitude

of the mutation in these cases has to be smaller than for parameters with a wider range. Using the same standard deviation for both types of parameters would result in the parameters that are restricted to the smaller range being zero or one most of the time, because the magnitude of the added mutation value would often exceed the range of the parameters.

In the present work, the parent and offspring populations are both set to contain two individuals, and one of those parents is an elitist. The standard deviation of the distributions used for the mutation are set as $\sigma = 2.0$ and $\sigma_{[0,1]} = 0.1$ respectively.

2. The parent population is initialized. This entails creating a set of individuals and randomly setting the values of their genomes. The range of these initial random values is limited to prevent excessive starting conditions. This is necessary because in preliminary experiments unrestricted random initial values often allowed for the creation of millions of nodes in a network, resulting in long calculation times when evaluating the fitness of an individual.

3. After the parent population has been initialized, a Growing Neural Gas network is created for each individual in the parent population, and its parameters are set to the parameter values encoded in that individual. Once created and initialized, the Growing Neural Gas networks are trained using the input samples from the training data set (see section 4.4), and a fitness value is assigned to each of them (the methods for the calculation of the fitness value are described in section 4.5).

4. The individuals of the offspring population are generated from the current parent population, using a crossover between two randomly selected parents as it is described in section 3.3.2.

5. The new offspring individuals are mutated by adding a random number $x \sim N(0, \sigma)$ to each parameter value in the individual's genome (a different deviation $\sigma_{[0,1]}$ is used for mutating the values of parameters that are restricted to the range $[0, 1]$). This randomization can result in parameter values that are no longer in their specified range (e.g., an error decay greater than one, with the error decay parameter being restricted to the range $(0, 1]$). In these cases, it is not sufficient to restrict the value that is used by the Growing Neural Gas algorithm to the according range and leave the value in the individual's genome unchanged, because this would allow the individual's values to continue evolving out of the parameter range while still possibly assigning

higher fitness values to it than to other individuals. Thus, whenever a parameter value exceeds the range specified for it, it is changed in the individual's genome before the parameters are used in the Growing Neural Gas algorithm for calculating the fitness value. Following the example of the error decay having a value greater than one, the value would be set to exactly one.

6. The fitness of the new offspring individuals evaluated as it is described in step 3.

7. The parents of the next generation are selected from all the populations of the current generation (i.e., the current parent and offspring individuals) using the elitist selection mechanism described in section 3.3.1.

8. The steps starting from 4 onwards are repeated until either a predetermined amount of generations or an individual with a fitness value greater than a preset value is found.

4.4 The Data Set

The training and test data sets used for the experiments described in section 5 are provided by Guillermo S. Donatti (unpublished data, 2008). Both are composed of vectors of Gabor Jets[9] which represent the features extracted from ETH-80 [Leibe and Schiele, 2003] object images. This database contains images of objects from four abstract categories (e.g., animals, fruits and vegetables) and eight concrete categories (e.g., horses, apples). Each concrete category includes 10 different objects represented by 41 images from viewpoints sampled equally over the upper viewing hemisphere. The data sets used in the present work contain features extracted from 256 different object images obtained from 4 objects of each concrete category represented by 8 of the 41 available view points.

This data is used for training and testing a feature-based object recognition and categorization model introduced by Guillermo S. Donatti (manuscript in preparation, 2008). This model is defined by a taxonomic hierarchy of self-organized Growing Neural Gas networks, which comprises three levels of knowledge organization: universal, abstract and concrete. During its training stage, features extracted from different object images and a distance

[9]Gabor Jets[Lades et al., 1993] are a collection of responses of *Gabor Wavelets*. Gabor wavelets are widely used in object recognition and computer vision in general, because they model the responses of V1 simple cells in the brain [Jones and Palmer, 1987].

measure are used to create an associative structure at each level, which represents the learned visual knowledge of its enclosed categories. During its recall stage, features extracted from test object images are matched, according to the same distance measure, against the learned object models, starting with the most abstract level of the hierarchy (i.e., the universal level) and descending to the more specific levels. During this process, the inference probabilities calculated on more abstract levels determine the ones of the more specific levels. Once the most specific level is reached, the object model with the highest probability is selected and the test object is finally identified.

The distance measure used both during the training and test stages is the normalized scalar distance, which is calculated using to the following equation:

$$d_{sn}(v_1, v_2) := 1 - \frac{\langle v_1 | v_2 \rangle}{||v_1|| \cdot ||v_2||} \tag{4.1}$$

where d_{sn} represents the dissimilarity[10] of two given vectors $v_1, v_2 \in \mathbb{R}_+^k$ and is in the range $[0, 1]$.

4.5 The Fitness Functions

An important part of the optimization process is the definition of a fitness function, which determines what kind of individuals are preferred[11] by the optimizer. In the present work, determining the fitness value of an individual is a process with the following structure: Let p be a vector containing the values of the parameters of the Growing Neural Gas algorithm stored in the individual for which the fitness value is to be calculated. First, the parameter set p is assigned to a Growing Neural Gas algorithm; the algorithm is initialized as described in section 2.7. Second, the network is trained using the Growing Neural Gas algorithm. The network's fitness value is calculated using the global error or the sample distance fitness function, both of which are described in the following sections.

4.5.1 Global Error

The fitness function presented here uses the global network error, which is the sum of the accumulated error values of all nodes after the network has

[10]This function (or rather the similarity function $S(v_1, v_2) = 1 - d_{sn}(v_1, v_2)$) is also used as an error measure in object recognition for bunchgraph matching and therefore it is suited for the data set described above.

[11]Which is either a minimum or a maximum value of the fitness function, depending on the definition of the fitness function: if the fitness value is an error, then the Evolutionary Algorithm is set to search for a minimum, and it is set to search for a maximum otherwise.

been trained:

$$f_G(p) := \sum_{i=0}^{N(p)} e_i(p), \tag{4.2}$$

where $e_i(p)$ is the accumulated error of the node i when training the network with Growing Neural Gas algorithm using the parameter set p, and $N(p)$ is the number of nodes in the trained network.

The error decay factors (i.e., α_{growth} and α_{error}, see section 2.3.1) are not optimized by the global error fitness function, because preliminary experiments suggest that they evolve to zero values, or to the smallest value permitted if they are restrained to values greater than zero. Thus, the global error is reduced to a minimum, independently of the quality (i.e., the capability of a network created with these parameters for learning the topology of the input manifold) of the parameter values. Therefore, in experiments that use the global error as the fitness function, the decay values are empirically set and are not optimized.

4.5.2 Sample Distance

Since the global network error cannot be used to optimize the error decay parameters, an alternative fitness function that is able to do it is desired. As both, α_{error} and α_{growth} are tied to the Growing Neural Gas algorithm's global error calculation, a fitness function independent of the algorithm is proposed. After the network is trained, a test set $S = \{s \in \mathbb{R}^{d_{in}}\}$ is used to calculate the distance of the network's nodes to the samples in S, according to the distance measure $d(x, y)$. The sum of these distances is used as the fitness value o the network:

$$f_{SD}(p) = \sum_{s \in S} d(x_{i_{0_s}}(p), s), \tag{4.3}$$

where $x_i(p)$ is the node of the network trained with the parameter set p, and i_{0_s} is the index of the node closest to the sample s:

$$i_{0_s} := \underset{j}{argmin}\, d(x_j(p), s) \tag{4.4}$$

4.5.3 Restricting Neuron Growth

It is possible to minimize the value of the sample distance function by filling the input space with a very high number of nodes. Since the node density resulting from this is very high, This could lead to a network that has one node for each test sample, which is not a desirable outcome, because the

network should extract the topology of the distribution and generalize over the input samples. Therefore, it is necessary to modify this fitness function to evaluate the networks proportionately to their number of nodes.

In the Growing Neural Gas algorithm the network's size is regulated with the utility measure (see section 2.4). However, this value is dependent on the parameter κ, which is included in the parameter values that are optimized and thus cannot be used in a fitness function.

A possible solution could be to calculate the density of the network and using it as its fitness value. However, the complexity of this approach exceeds the scope of the present work and therefore is not developed further. Instead, a less complex solution is proposed and described in section 4.5.4.

4.5.4 Sample Distance with Restricted Growth

The fitness function proposed in this section is a modified version of the one described in section 4.5.2. In addition to calculating the sample distance, it "punishes" individuals by adding one to their fitness value for each node contained in the networks trained with the individuals' parameter values. Thus, the fitness value is calculated according to the following equation:

$$f_{SD+1}(p) = \sum_{s \in S} d(x_{i_{0_s}}(p), s) + N(p) = f_{SD}(p) + N(p) \tag{4.5}$$

where $N(p)$ is the number of nodes after training a network with the parameter values p. This function has small values when the number of nodes is small and the distance of the nodes to the test input samples is low.

5 Discussion

5.1 Multiple presentations

The optimization process described in section 4.3 is used to optimize the global error fitness function. During the evaluation of the individuals in the Evolutionary Algorithm, Growing Neural Gas networks are trained with input samples from 22% of the training data set.

The fitness values calculated during the optimization process do not provide sufficient information for evaluating the optimization. Therefore, the optimized parameter values are used for training a separate Growing Neural Gas network with the same data set. The global and mean errors recorded during training of this network are shown in Figure 1. They display rela-

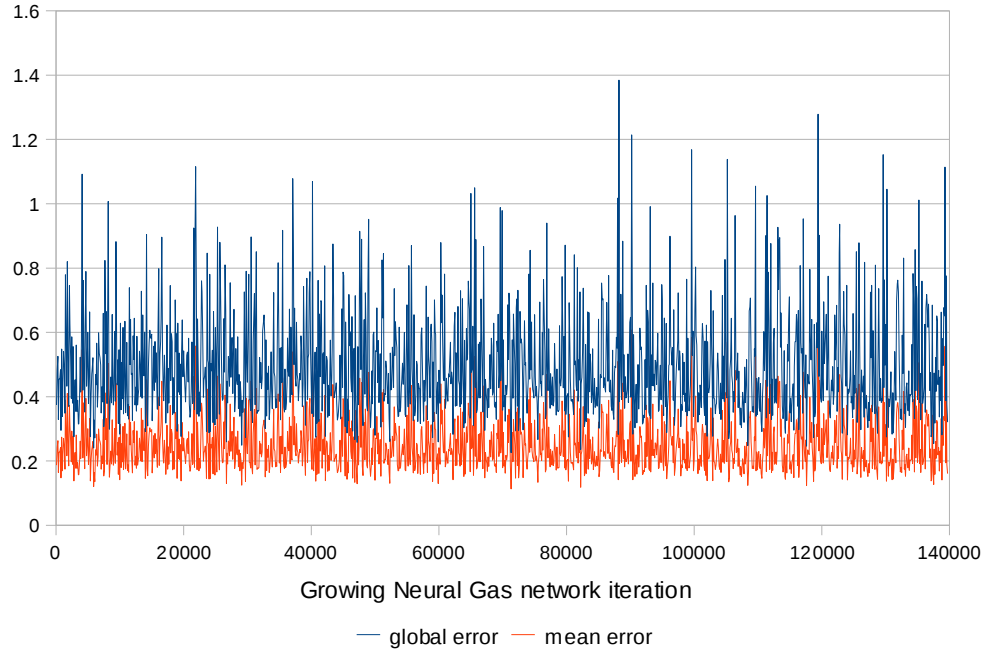

Figure 1: Training a Growing Neural Gas network with the parameters from an optimization with the global error fitness function results in these error curves. The global and mean errors oscillate in the range $[0.16, 1.4]$, indicating that no learning occurs during the training of the network.

tively low error values in all iterations. However, both, the global and mean error oscillate between the same values (approximately 0.16 and 1.4) without showing any tendency towards being decreased. This indicates that networks trained with these parameter values are unable to learn the topology of the input manifold.

The values of both, the decay and growth generation length parameters, are set to one. This results in the creation of a new node and the deletion of a useless one for every input sample presented to the Growing Neural Gas network during its training (see section 2.7, step 11 and 12). Additionally, the parameters controlling the movement of the winner node and its neighbors are set to one as well, hence the nodes with a low distance to the input samples presented to the Growing Neural Gas network are moved strongly towards them. While these optimized values minimize the global and mean errors, they also lead to the undesired behavior described above.

One possible reason for this undesired behavior is the relatively short training time during the optimization. The input samples for each network

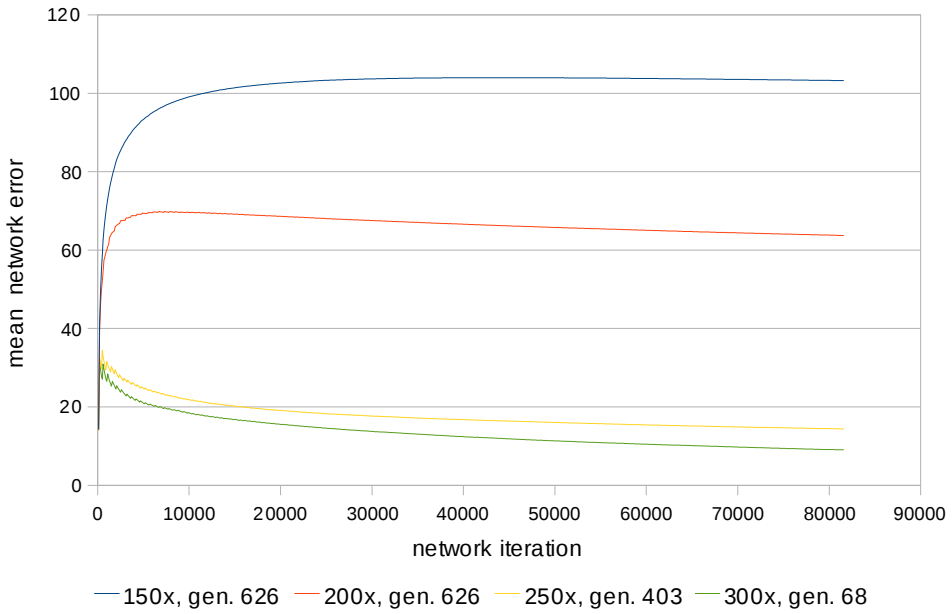

Figure 2: The effect of multiple presentations. Networks to which training samples are presented 300 times learn significantly better than the ones trained with 150 presentations. Additionally, the results are found earlier (at generation 68 as opposed to generation 403). The parameter values for these networks are evolved with 11% of the training data set and the sample distance fitness function.

are extracted from 22% of the training data set, thus the networks are trained with approximately 1000 input samples. Only a small amount of learning can occur during the presentation of these input samples, thus networks that show the undesired behavior described above would be preferred by the Evolutionary Algorithm, because networks capable of learning usually show a relatively high error during early iterations.

To solve this problem, training input samples are presented repeatedly during the evaluation of the Growing Neural Gas networks. Empirical results (see Figure 2) suggest that presenting the input samples 300 times achieves low error values while the calculation times for evaluating individuals are still relatively short.

5.1.1 Reducing Calculation Time

Presenting samples repeatedly significantly increases the time required for evaluating an individual during the optimization process. Hence, empirically

determined parameter values provided by Guillermo S. Donatti (unpublished data, 2008) are used as initial values for the individuals in the first parent population. These parameter values are known to produce low global and mean network errors and are used during the training of the feature-based object recognition and categorization model described in section 4.4.

This approach significantly limits the size of the region of the search space the Evolutionary Algorithm has to explore, because the individuals already start out near parameter values known to produce networks that show the desired behavior. Additionally, every individual fitter than the one containing the empirically determined parameters should improve upon them, if the fitness functions work as intended.

5.1.2 The Covariance Matrix Adaptation

An additional approach to reducing the time for finding an optimum is using the Covariance Matrix Adaptation (see section 3.4) instead of the Evolutionary Algorithm described in section 4.3, because using this strategy decreases the amount of generations necessary for finding optimal parameter values. However, the CMA strategy is only designed for optimizing problems with real valued parameters. Furthermore, experiments using the CMA strategy for optimizing Growing Neural Gas parameter values suggest that it cannot adapt parameters with integral values. This is most likely caused by the non-steady nature of integral values (the parameter values with integral dimensions are truncated, i.e., rounded down for use in the Growing Neural Gas algorithm when the CMA strategy is used). However, further research on this topic is needed, since the CMA strategy could significantly reduce the time required for finding optimal parameter values.

5.2 The Experiments

The algorithm described in section 4.3 is used to optimize parameter values for 22% of the training data set. The approaches discussed in sections 5.1–5.1.2 are taken into account as well: input samples from the training data set are presented 300 times, and the values of the genomes of the individuals in the parent population of the first generation are constrained to the empirically set parameter values for the Growing Neural Gas algorithm. The resulting optimized parameter values are then used to calculate the different fitness values for a separate Growing Neural Gas network trained with the same data set that is used during the optimization. Experiments with this setup are conducted for each of the fitness functions introduced in section 4.5, and the results are discussed in the following sections.

5.2.1 The Global Error Fitness Function

The results obtained when using the global error fitness function are presented in Figure 3. They show, that the Growing Neural Gas networks created with the optimized parameter values generate a lower global network error than the ones created with the empirically set parameters. However, the speed of learning (i.e., the decrease of the error) is not optimized by the fitness function: the global error of the network from generation 114 indicates a much slower decrease than the one of the baseline Growing Neural Gas network[12], but at the same time arrives at a lower error.

The values of the f_{SD+1} function of the empirically set parameters and the individual from generation 114 behave similarly, while the fitness values of the winner individual display a much higher increase in the final iterations, which is likely caused by a greater growth of the Growing Neural Gas network[13]. These results support the idea of needing a growth restriction for the sample distance fitness function introduced in section 4.5.4.

5.2.2 The Sample Distance Fitness Function

The results for parameters extracted from individuals obtained with an optimization process using the f_{SD} fitness function show that the optimization does not reduce the global error (see Figure 4). This may be caused by the fact that the optimized error decay factors (α_{growth} and α_{error}, see section 2.7, step 13) are both set to one, suggesting that once a Growing Neural Gas network trained with these values accumulates an error, it can only lower its global error by removing a node. The decay parameter values probably cause the near-steady (i.e., almost steady with only small reductions) increase of the global error that can be observed in Figure 4. Furthermore, it is possible that the Growing Neural Gas networks trained with the optimized parameter values retain a high global error due to the values of the decay factors and still learn the topology of the input manifold, but this cannot be evaluated using the error measures of the Growing Neural Gas algorithm. However, using the optimized parameter values may show increased object recognition and categorization rates with the model introduced in section 5.4.

It is also worth noting that the networks that use the optimized parameter values seem to have a tendency to constantly increase the number of their nodes, as already observed with the global error fitness function in section

[12]The baseline Growing Neural Gas network uses the empirically set parameter values as described in section 5.1.1.

[13]This can be deduced from the fact that $f_{SD+1} - f_{SD} = N$. Since the values for f_{SD} of all individuals only have a small difference, the increase in f_{SD+1} results mostly from additional nodes in the network.

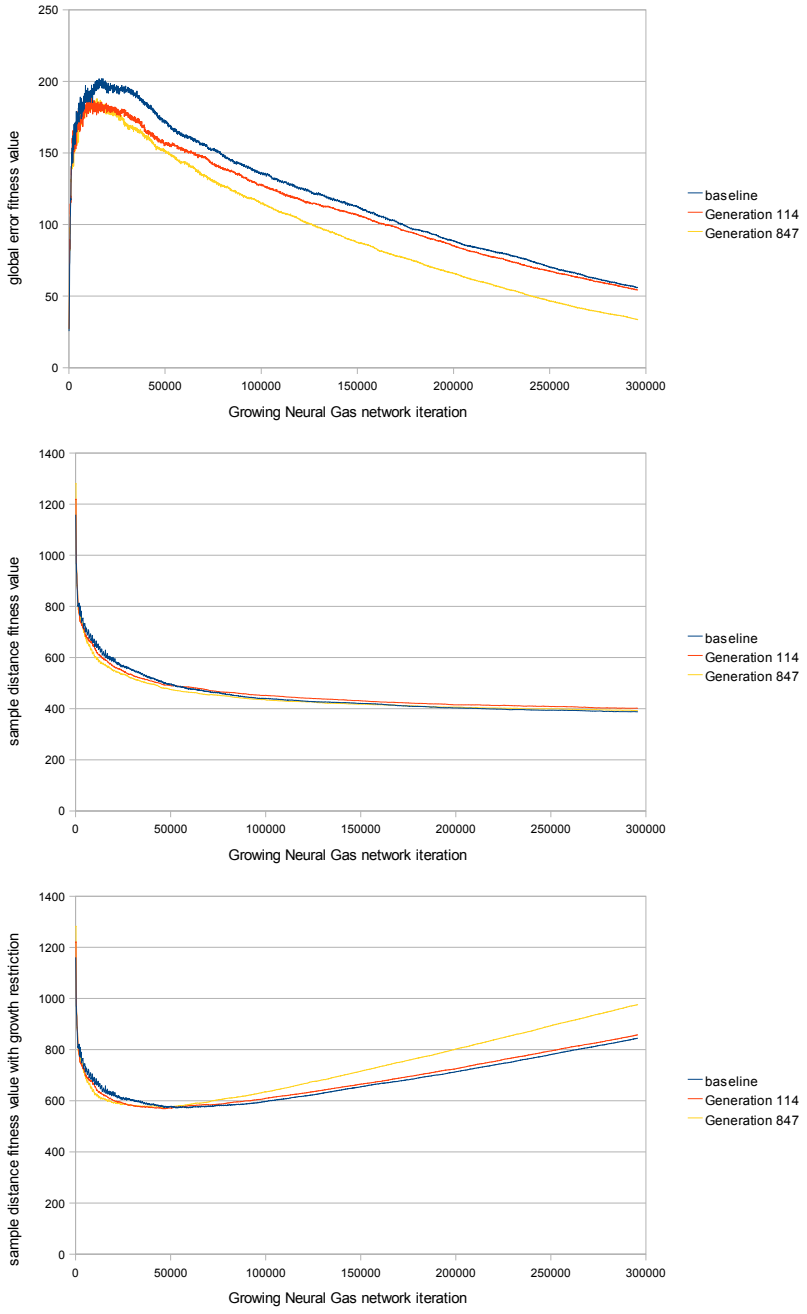

Figure 3: The graphs show the three fitness functions calculated for a Growing Neural Gas network trained with parameter values obtained from an optimization process using the global error fitness function and the method described in section 5.2.

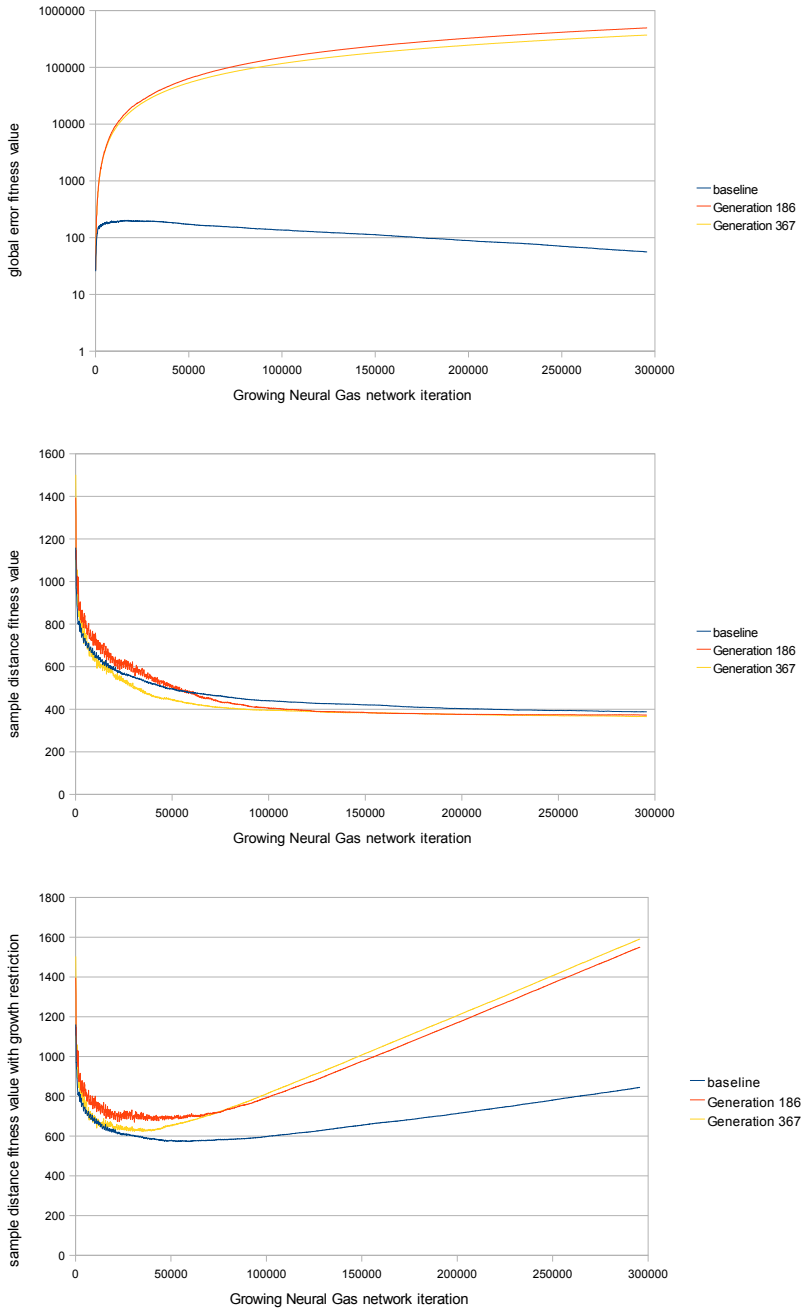

Figure 4: The graphs show the three fitness functions calculated for Growing Neural Gas networks using parameter values that were optimized with the sample distance measure and the methods described in section 5.2.

5.2.1. In general, longer evolutions would probably lead to networks with a high number of nodes, as it is described in section 4.5.3.

5.2.3 Sample Distance with Restricted Growth

The results for using the f_{SD+1} fitness function for the Evolutionary Algorithm are presented in Figure 5. They suggest that the goal of restricting the amount of nodes in the Growing Neural Gas networks is achieved: while the difference between the sample distance values of all networks is relatively small, the values of f_{SD+1} decrease significantly compared to the baseline values, indicating a reduction in the amount of nodes created by the network.

The fittest individual found during the optimization process shows a global error that increases almost steadily, likely caused by the same reasons described in section 5.2.2. However, the sample distance values of all individuals (see Figure 5) show only a small difference between each other, hence it is possible that the network correctly learns the topology. This hypothesis is also supported by the fact that a Growing Neural Gas network trained with parameter values extracted from an individual of generation 187, which contains decay values less than one, shows lower global error values than the network that is trained using the empirically set parameter values.

5.3 Random Initialization

Additional experiments are conducted using the same setup as described in section 5.2, but with random initial values for the individuals in the parent population of the first generation. The results[14] for these experiments are presented in Figure 6.

In general, most of the Growing Neural Gas networks trained with the parameter values obtained have a low initial error. However, they do not show any learning behavior since the error oscillates within a fixed range. Although networks with parameter values from subsequent generations seem to lower the limits of this range, they do not show a tendency towards learning either. These solutions are probably preferred by the optimizer because they are near a local optimum.

These local optima could probably be avoided by running the optimization process multiple times, until the individuals in the first parent population start near a region of the search space that describes networks that show learning behavior. However, directly evaluating learning capabilities is preferable, as it would not rely on such random influences.

[14]These results can only be considered preliminary, as more time is needed to properly evolve parameter sets with this method.

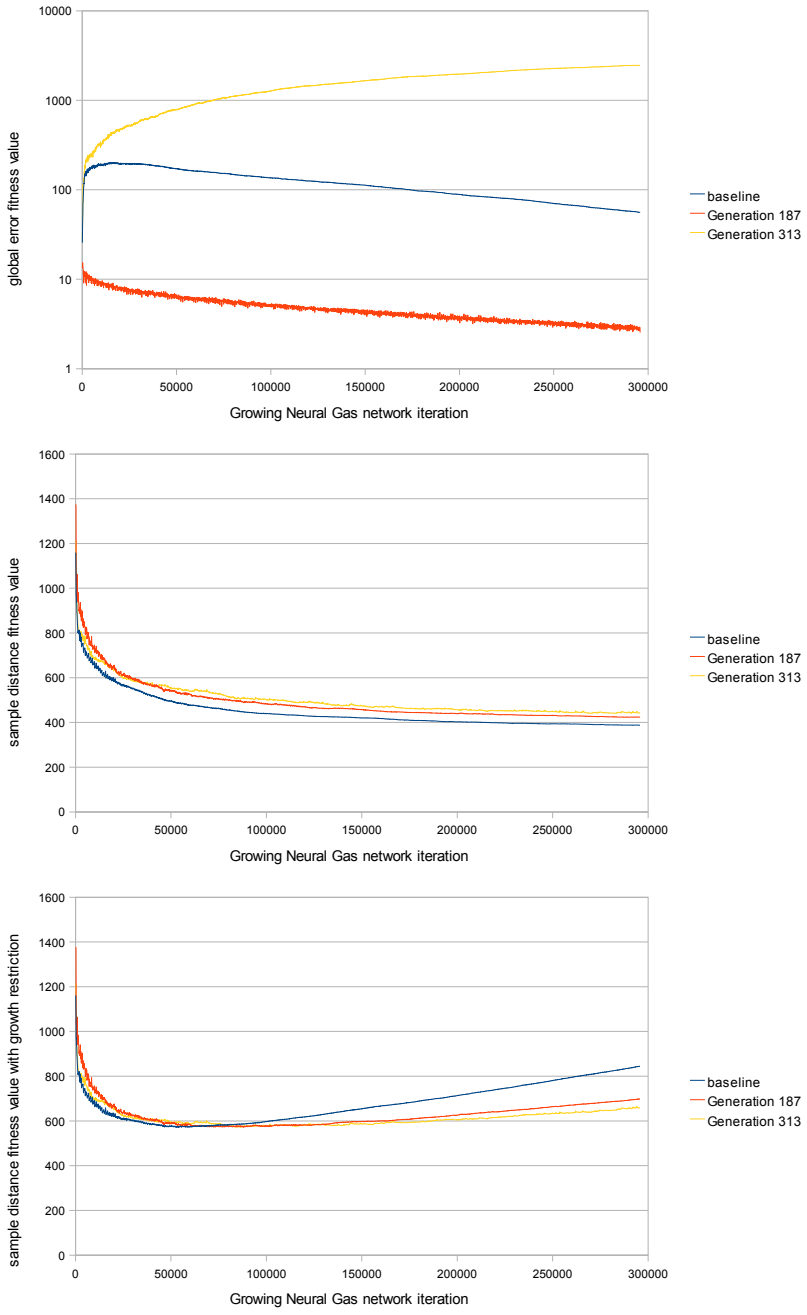

Figure 5: These diagrams show the three fitness functions calculated for Growing Neural Gas networks using parameter values optimized with the sample distance fitness function with restricted growth (the optimization process described in section 5.2 is used).

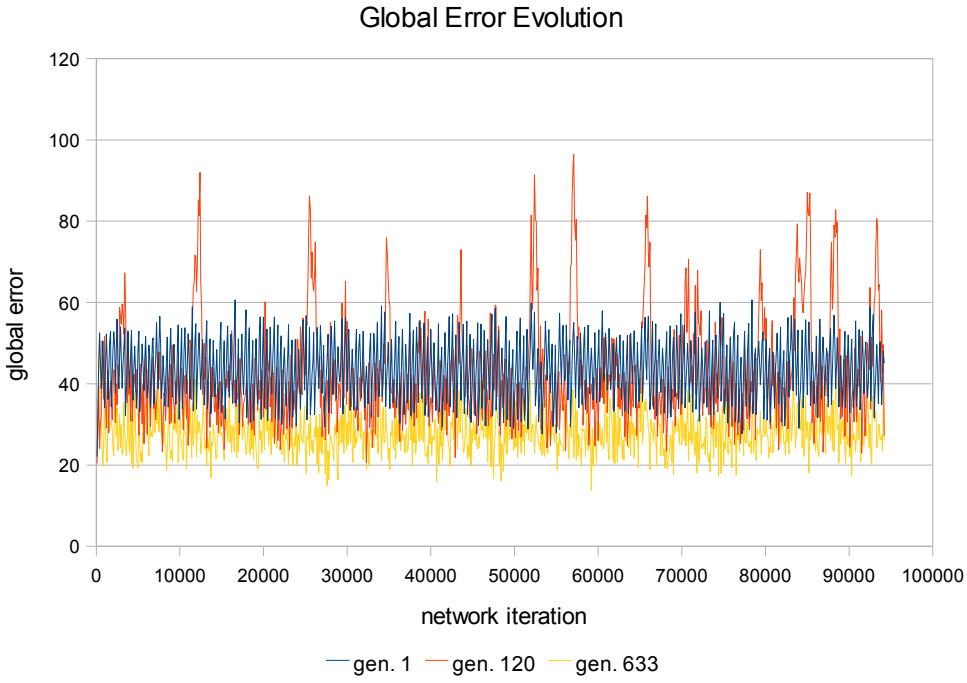

Figure 6: Global error curves of networks trained with parameter sets obtained from three generations of the optimization process using the global error fitness function and starting from random parameter sets.

In a future work, additional experiments with random initializations need to be done. The effect of the restriction of the initial values of the individuals in the first parent population (see section 4.3) on the quality of the solutions (i.e., the capability of networks trained with the optimized paramete values) needs to be investigated as well.

5.4 Comparing the Fitness Functions

In the present work, three different fitness functions are proposed to find optimal individuals. Their genomes are used to set the parameter values of the Growing Neural Gas algorithm (see section 4.3) and the obtained results are presented in section 5.2. However, the performances cannot be cross-compared directly. Furthermore, the global or mean error values of Growing Neural Gas networks trained with the optimized parameter values cannot be used to perform this task either, since they depend on the error decay parameters.

In order to cross-compare the performance of these parameter values,

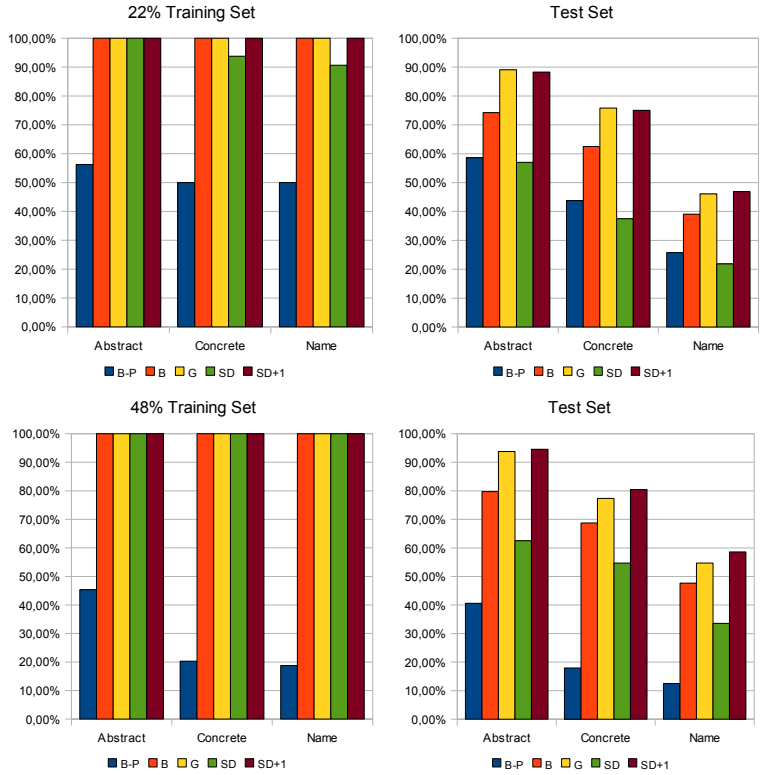

Figure 7: Recognition and categorization rates achieved with the feature-based categorization and object recognition model for 22% and 48% of the data set.
The abbreviations are: B – Baseline, B-P – Baseline without plasticity, G – Global Error, SD – Sample distance, SD+1 – Sample distance with growth restriction.

the different object recognition and categorization rates obtained from the model introduced in section 4.4 are used, and the model's underlying Growing Neural Gas taxonomy is trained with the optimized parameter values. Additionally, the object recognition and categorization rates can be used to indicate the practical effect of the parameter value optimization.

The resulting experimental set is composed of five instances of the model, one for each parameter set obtained using different fitness functions, one for the empirically determined parameter sets, and another one for a parameter set which uses a combination between the empirically determined parameter values and a plasticity function[15].

[15]The plasticity function is an empirically set function which regulates the node growth

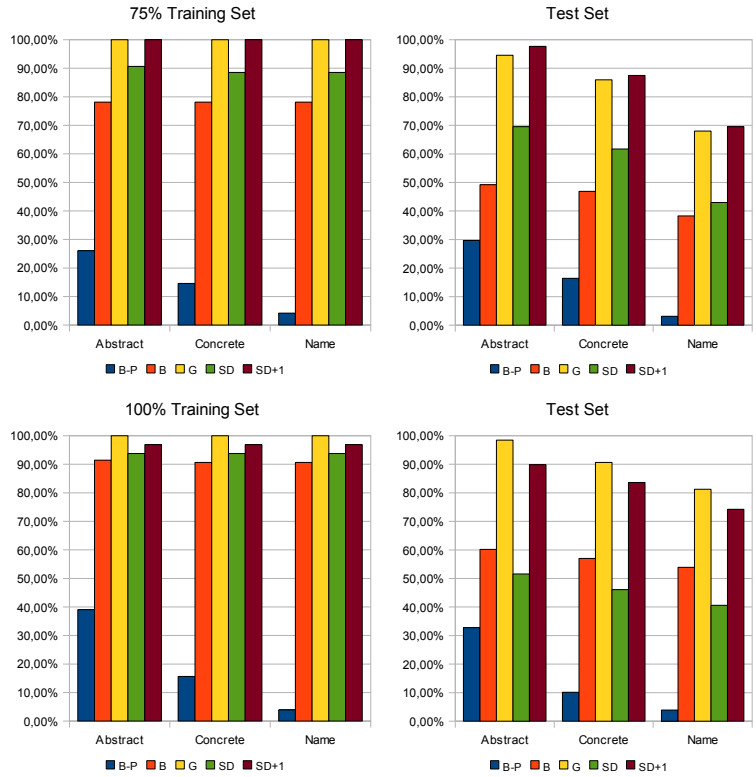

Figure 8: Recognition and categorization rates achieved with the feature-based categorization and object recognition model for 75% and 100% of the data set.
The abbreviations are: B – Baseline, B-P – Baseline without plasticity, G – Global Error, SD – Sample distance, SD+1 – Sample distance with growth restriction.

The percentage of object images correctly categorized (abstract and concrete) and recognized (object names) achieved by the models both for the training and test data sets introduced in section 4.4 are presented in Figure 7 and 8. The results show, that the models trained with the parameters optimized with the global error and sample distance fitness function with growth restriction achieve significantly higher object recognition and categorization rates than the models trained with the baseline parameter values with the plasticity function. However, the models trained with the sample distance function without the growth restriction show lower object recognition and

during training of a Growing Neural Gas network based on its growth and decay factor parameters and the size of the training data set that is used.

categorization rates than the model trained with empirically determined values.

In general, the object recognition and categorization rates seem to stay relatively constant when more input samples are presented. On the other hand, the ones of the model trained with empirically set parameter values with the plasticity function seem to decrease proportionately to the number of samples presented.

In all cases, the optimized parameter values outperform the empirically determined ones.

6 Conclusion and Further Research

The results presented in section 5.4 indicate that it is possible to find parameter sets, which improve the object recognition and categorization rates of the model described in section 4.4, when starting the optimization process with constrained parameter values and using the global error fitness function. Furthermore, similar improvements can be obtained when using the sample distance fitness function with the growth restriction. However, the general goal of finding optimal parameter values starting the optimization process from arbitrary values could not be reached.

There are two obstacles that need to be overcome in future research: one is the long calculation time needed for training Growing Neural Gas networks during the evaluation of parameter values; the other is avoiding the apparently high number of local optima, leading to networks with very low error values, but also with little capability of learning the distribution of the input samples. A possible approach to overcome this can be to modify the fitness functions to take the learning capabilities of a network into account (e.g., by assigning Growing Neural Gas networks a lower fitness value for errors generated in later generations and disregarding errors in earlier ones). Another possible solution can be incorporating a rate of change in the error curve into a fitness function.

Additionally, reducing the number of generations necessary for finding an optimum is an important task for future research, because of the time required for evaluating the individual in each generation. Therefore, a method for using the Covariance Matrix Adaptation for integral dimensions is desired. This might also allow for the usage of the Covariance Matrix Adaptation for Multi-objective Optimization[Igel et al., 2007] to restrict the growth rates of the Growing Neural Gas networks.

The growth restriction placed on the sample distance fitness function may be too restrictive, favoring small networks over networks that perform the

desired quantization. In a future work, the possibility of allowing for gradual restriction of network size and the distance from the test input samples could be examined.

Further experiments using randomly initialized individuals in the first parent population are needed to determine the suitability of this method. The effect of the restrictions placed on the ranges of the random values for the initial parent population needs to be investigated as well.

References

Beyer, G. (2007). Evolution strategies. *Scholarpedia*, 2(8):1965.

Bolder, B. (2005). *Coordination of an Artificial Visual System with Biological Models*. PhD thesis, Ruhr-Universitt Bochum.

Fritzke, B. (1995). A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems*. MIT Press, Cambridge MA.

Fritzke, B. (1997). A self-organizing network that can follow non-stationary distributions. Proceedings of ICANN-97, International Conference on Artificial Neural Networks, pages 613–618. Springer.

Igel, C., Hansen, N., and Roth, S. (2007). Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15(1):1–28.

Igel, C., Suttorp, T., and Hansen, N. (2006). A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 453–460. ACM Press.

Jones, J. and Palmer, L. (1987). An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1233–1258.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, (43):56–69.

Kreutz, M., Sendhoff, B., and Igel, C. (2008). Ealib: A c++ class library for evolutionary algorithms.

Lades, M., Vorbrüggen, J. C., Buhmann, J. M., Lange, J., von der Malsburg, C., Würtz, R. P., and Konen, W. (1993). Distortion invariant object

recognition in the dynamic link architecture. *IEEE Trans. Computers*, 42(3):300–311.

Leibe, B. and Schiele, B. (2003). Analyzing appearance and contour based methods for object categorization. In *CVPR (2)*, pages 409–415.

Martinetz, T. and Schulten, K. (1991). A "neural-gas" network learns topologies. In Kohonen, T. M., K., S., O., and Kangas, J., editors, *Artificial Neural Networks*, pages 397–402. North-Holland, Amsterdam.

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

Datum

Unterschrift