Efficient Approximation of a Recursive Growing Neural Gas

Jochen Kerdels[1] and Gabriele Peters[1]

FernUniversität in Hagen – University of Hagen,
Human-Computer Interaction, Faculty of Mathematics and Computer Science,
Universitätsstrasse 1 , D-58097 Hagen, Germany
{jochen.kerdels, gabriele.peters}@fernuni-hagen.de

Abstract. The recursive growing neural gas algorithm (RGNG) is a variant of the classic GNG that was specifically designed to model the response behavior of groups of biological neurons. It was used successfully to describe the behavior of entorhinal grid cells as well as entorhinal cells that show grid like activity in response to saccadic eye movements. More recently, the RGNG algorithm was integrated into a model of cortical column function as part of an autoassociative memory cell. To facilitate future research involving the simulation of hundreds to thousands of neuron groups we present an alternative algorithm to the RGNG as a drop-in replacement in the context of neuron group modeling. The *differential growing neural gas* (DGNG) is structurally less complex, more efficient to compute, and more robust in terms of the input space representation that is learned while retaining most of the RGNG's important characteristics. We provide a formal definition of the DGNG algorithm and demonstrate its characteristics with a first set of experiments.

Keywords: Recursive Growing Neural Gas · Differential Growing Neural Gas · Representation Learning · Modeling of Neuron Groups.

1 Introduction

In a recent paper [12] we outlined a functional model of cortical columns [19, 18, 1] that utilized a *recursive growing neural gas* (RGNG) as one of its core components. More specifically, two reciprocally coupled RGNGs were used to describe two groups of neurons within a single cortical column that together form a local, autoassociative memory cell (AMC). Originally, we introduced the RGNG algorithm [8, 6] to model the behavior of entorhinal grid cells [3, 4]. In this earlier model a single RGNG was used to describe a single group of neurons, effectively modeling one half of the later proposed cortical AMC.

Here we review the RGNG algorithm and introduce an algorithmic alternative that is structurally less complex, more efficient to compute, and more robust in terms of the input space representation that is learned by the algorithm. Given that our future research on cortical column models aims at simulating networks of hundreds to thousands of cortical columns it seems prudent to improve and

optimize the model used so far in light of this new field of application, i.e., when moving away from simulating single groups of neurons towards simulating entire networks of neuron groups.

The next two sections provides a recapitulation of the RGNG algorithm highlighting its core characteristics and its usage in the context of modeling the response behavior of a group of neurons. Section 3 introduces a novel algorithm, the *differential growing neural gas*, that addresses some of the shortcomings of the RGNG while maintaining its main properties in the context of modeling neuron groups. In section 4 the behavior of the introduced algorithm is demonstrated and analyzed using the well known MNIST dataset [16]. Finally, section 5 concludes the paper and outlines aspects of future research.

2 RGNG Revisited

The recursive growing neural gas (RGNG) is an unsupervised learning algorithm that learns a prototype-based representation of a given input space. Although the RGNG is algorithmically similar to well known prototype-based methods of unsupervised learning like the original growing neural gas (GNG) [17, 2] or self organizing maps [15], the resulting input space representation is significantly different from common prototype-based approaches. While the latter use single prototype vectors to represent local regions of input space that are pairwise disjoint, the RGNG uses a sparse distributed representation where each point in input space is encoded by a joint ensemble activity.

From a neurobiological perspective an RGNG can be interpreted as describing the behavior of a group of neurons that receives signals from a common input space. Each of the neurons in this group tries to learn a coarse representation of the *entire* input space while being in competition with one another. This coarse representation consists of a limited number of prototypical input patterns that are learned competitively and stored on separate branches of the neuron's dendritic tree. The competitive character of the learning process ensures that the prototypical input patterns are distributed across the entire input space and thus form a coarse, pointwise representation of it. In addition to these intra-neuronal processes the inter-neuronal competition on the neuron group level influences the alignment of the individual neuron's representations. More specifically, the competition between the neurons favors an alignment of the individual representations in such a way that the different representations become pairwise distinct. As a result, the neuron group as a whole forms a dense representation of the input space consisting of the self-similar, coarse representations of its group members. The activity of individual neurons in such a group in response to a given input is ambiguous as it cannot be determined from the outside which of the learned input patterns triggered the neuron's response. However, the collective activity of all neurons in response to a shared input creates an activity pattern that is highly specific for the given input since it is unlikely that for two different inputs the response of all neurons would be exactly the same.

A formal description of the RGNG algorithm is given in the next section. It was adapted from [9]. The given description is independent of the RGNG's application in the aforementioned neurobiological context, which is described later in section 2.2.

2.1 Formal Description

An RGNG g can be described by a tuple[1]:

$$g := (U, C, \theta) \in G,$$

with a set U of units, a set C of edges, and a set θ of parameters. Each unit u is described by a tuple:

$$u := (w, e) \in U, \quad w \in W := \mathbb{R}^n \cup G, \;\; e \in \mathbb{R},$$

with the *prototype* w, and the *accumulated error* e. Note that the prototype w of an RGNG unit can either be a n-dimensional vector or another RGNG. Each edge c is described by a tuple:

$$c := (V, t) \in C, \quad V \subseteq U \;\wedge\; |V| = 2, \;\; t \in \mathbb{N},$$

with the units $v \in V$ connected by the edge and the *age* t of the edge. The *direct neighborhood* E_u of a unit $u \in U$ is defined as:

$$E_u := \{k | \exists\, (V, t) \in C, \;\; V = \{u, k\}\,, \;\; t \in \mathbb{N}\}\,.$$

The set θ of parameters consists of:

$$\theta := \{\epsilon_b, \epsilon_n, \epsilon_r, \lambda, \tau, \alpha, \beta, M\}\,.$$

The behavior of an RGNG is defined by four functions. The distance function

$$D(x, y) : W \times W \to \mathbb{R}$$

determines the distance either between two vectors, two RGNGs, or a vector and an RGNG. The interpolation function

$$I(x, y) : (\mathbb{R}^n \times \mathbb{R}^n) \cup (G \times G) \to W$$

generates a new vector or new RGNG by interpolating between two vectors or two RGNGs, respectively. The adaptation function

$$A(x, \xi, r) : W \times \mathbb{R}^n \times \mathbb{R} \to W$$

adapts either a vector or RGNG towards the input vector ξ by a given fraction r. Finally, the input function

$$F(g, \xi) : G \times \mathbb{R}^n \to G \times \mathbb{R}$$

feeds an input vector ξ into the RGNG g and returns the modified RGNG as well as the distance between ξ and the best matching unit s_1 (BMU, see below) of g. The input function F contains the core of the RGNG's behavior and utilizes the other three functions, but is also used, in turn, by those functions introducing several recursive paths to the program flow.

[1] The notation $g.\alpha$ is used to reference the element α within the tuple.

$\boldsymbol{F(g, \xi)}$*:* The input function F is a generalized version of the original GNG algorithm that facilitates the use of prototypes other than vectors. In particular, it allows to use RGNGs themselves as prototypes resulting in a recursive structure. An input $\xi \in \mathbb{R}^n$ to the RGNG g is processed by the input function F as follows:

- Find the two units s_1 and s_2 with the smallest distance to the input ξ according to the distance function D:

$$s_1 := \arg\min_{u \in g.U} D(u.w, \xi)\,,$$
$$s_2 := \arg\min_{u \in g.U \setminus \{s_1\}} D(u.w, \xi)\,.$$

- Increment the age of all edges connected to s_1:

$$\Delta c.t = 1, \quad c \in g.C \wedge s_1 \in c.V\ .$$

- If no edge between s_1 and s_2 exists, create one:

$$g.C \Leftarrow g.C \cup \{(\{s_1, s_2\}, 0)\}\,.$$

- Reset the age of the edge between s_1 and s_2 to zero:

$$c.t \Leftarrow 0, \quad c \in g.C \wedge s_1, s_2 \in c.V\ .$$

- Add the squared distance between ξ and the prototype of s_1 to the accumulated error of s_1:

$$\Delta s_1.e = D(s_1.w, \xi)^2\,.$$

- Adapt the prototype of s_1 and all prototypes of its direct neighbors:

$$s_1.w \Leftarrow A(s_1.w, \xi, g.\theta.\epsilon_b)\,,$$
$$s_n.w \Leftarrow A(s_n.w, \xi, g.\theta.\epsilon_n)\,, \forall s_n \in E_{s_1}.$$

- Remove all edges with an age above a given threshold τ and remove all units that no longer have any edges connected to them:

$$g.C \Leftarrow g.C \setminus \{c | c \in g.C \wedge c.t > g.\theta.\tau\}\,,$$
$$g.U \Leftarrow g.U \setminus \{u | u \in g.U \wedge E_u = \emptyset\}\,.$$

- If an integer-multiple of $g.\theta.\lambda$ inputs was presented to the RGNG g and $|g.U| < g.\theta.M$, add a new unit u. The new unit is inserted "between" the unit j with the largest accumulated error and the unit k with the largest accumulated error among the direct neighbors of j. Thus, the prototype $u.w$ of the new unit is initialized as:

$$u.w := I(j.w, k.w)\,, j = \arg\max_{l \in g.U} (l.e)\,,$$
$$k = \arg\max_{l \in E_j} (l.e)\,.$$

The existing edge between units j and k is removed and edges between units j and u as well as units u and k are added:

$$g.C \Leftarrow g.C \setminus \{c | c \in g.C \wedge j, k \in c.V\},$$
$$g.C \Leftarrow g.C \cup \{(\{j, u\}, 0), (\{u, k\}, 0)\}.$$

The accumulated errors of units j and k are decreased and the accumulated error $u.e$ of the new unit is set to the decreased accumulated error of unit j:

$$\Delta j.e = -g.\theta.\alpha \cdot j.e, \quad \Delta k.e = -g.\theta.\alpha \cdot k.e,$$
$$u.e := j.e\ .$$

– Finally, decrease the accumulated error of all units:

$$\Delta u.e = -g.\theta.\beta \cdot u.e, \quad \forall u \in g.U\ .$$

The function F returns the tuple $(g, d_{\min})$ containing the now updated RGNG g and the distance $d_{\min} := D(s_1.w, \xi)$ between the prototype of unit s_1 and input ξ. Note that in contrast to the regular GNG there is no stopping criterion any more, i.e., the RGNG operates explicitly in an online fashion by continuously integrating new inputs. To prevent unbounded growth of the RGNG the maximum number of units $\theta.M$ was introduced to the set of parameters.

D*(*x*, *y*):* The distance function D determines the distance between two prototypes x and y. The calculation of the actual distance depends on whether x and y are both vectors, a combination of vector and RGNG, or both RGNGs:

$$D(x, y) := \begin{cases} D_{RR}(x, y) \text{ if } x, y \in \mathbb{R}^n, \\ D_{GR}(x, y) \text{ if } x \in G \wedge y \in \mathbb{R}^n, \\ D_{RG}(x, y) \text{ if } x \in \mathbb{R}^n \wedge y \in G, \\ D_{GG}(x, y) \text{ if } x, y \in G. \end{cases}$$

In case the arguments of D are both vectors, the Minkowski distance is used:

$$D_{RR}(x, y) := \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}, \; x = (x_1, \ldots, x_n), \; y = (y_1, \ldots, y_n), \; p \in \mathbb{N}.$$

Using the Minkowski distance instead of the Euclidean distance allows to adjust the distance measure with respect to certain types of inputs via the parameter p. For example, setting p to higher values results in an emphasis of large changes in individual dimensions of the input vector versus changes that are distributed over many dimensions [7]. However, in the common case the parameter is set to a fixed value of 2 which makes the Minkowski distance equivalent to the Euclidean distance.

In case the arguments of D are a combination of vector and RGNG, the vector is fed into the RGNG using function F and the returned minimum distance is taken as distance value:

$$D_{GR}(x, y) := F(x, y)\text{.}d_{\min},$$
$$D_{RG}(x, y) := D_{GR}(y, x)\,.$$

In case the arguments of D are both RGNGs, the distance is defined to be the pairwise minimum distance between the prototypes of the RGNGs' units, i.e., *single linkage* distance between the sets of units is used:

$$D_{GG}(x, y) := \min_{u \in x\text{.}U,\, k \in y\text{.}U} D(u\text{.}w,\, k\text{.}w)\,.$$

The latter case is used by the interpolation function if the recursive depth of an RGNG is at least 2. In a neurobiological context (see 2.2) an RGNG-based model typically has only a recursive depth of 1. Hence, the case is considered for reasons of completeness rather than necessity. Alternative measures to consider could be, e.g., *average* or *complete* linkage.

I*(*x*, *y*):* The interpolation function I returns a new prototype as a result from interpolating between the prototypes x and y. The type of interpolation depends on whether the arguments are both vectors or both RGNGs:

$$I(x, y) := \begin{cases} I_{RR}(x, y) \text{ if } \; x, y \in \mathbb{R}^n, \\ I_{GG}(x, y) \text{ if } \; x, y \in G. \end{cases}$$

In case the arguments of I are both vectors, the resulting prototype is the arithmetic mean of the arguments:

$$I_{RR}(x, y) := \frac{x + y}{2}.$$

In case the arguments of I are both RGNGs, the resulting prototype is a new RGNG a. Assuming w.l.o.g. that $|x\text{.}U| \geq |y\text{.}U|$ the components of the interpolated RGNG a are defined as follows:

$$a := I(x, y)\,,$$
$$a\text{.}U := \left\{ (w, 0) \;\middle|\; \begin{array}{l} w = I(u\text{.}w, k\text{.}w)\,, \\ \quad \forall u \in x\text{.}U, \\ \quad k = \arg\min\limits_{l \in y\text{.}U} D(u\text{.}w,\, l\text{.}w) \end{array} \right\},$$
$$a\text{.}C := \left\{ (\{l, m\}, 0) \;\middle|\; \begin{array}{ll} \exists c \in x\text{.}C & \\ \wedge & u, k \in c\text{.}V \\ \wedge & l\text{.}w = I(u\text{.}w, \cdot) \\ \wedge & m\text{.}w = I(k\text{.}w, \cdot) \end{array} \right\},$$
$$a\text{.}\theta := x\text{.}\theta\,.$$

The resulting RGNG a has the same number of units as RGNG x. Each unit of a has a prototype that was interpolated between the prototype of the corresponding unit in x and the nearest prototype found in the units of y. The edges and parameters of a correspond to the edges and parameters of x.

A*(x, ξ, r):* The adaptation function A adapts a prototype x towards a vector ξ by a given fraction r. The type of adaptation depends on whether the given prototype is a vector or an RGNG:

$$A(x, \xi, r) := \begin{cases} A_R(x, \xi, r) \text{ if } x \in \mathbb{R}^n, \\ A_G(x, \xi, r) \text{ if } x \in G. \end{cases}$$

In case prototype x is a vector, the adaptation is performed as linear interpolation:

$$A_R(x, \xi, r) := (1 - r)\, x + r\, \xi.$$

In case prototype x is an RGNG, the adaptation is performed by feeding ξ into the RGNG. Importantly, the parameters ϵ_b and ϵ_n of the RGNG are temporarily changed to take the fraction r into account:

$$\begin{aligned} \theta^* &:= (\, r,\ \ r \cdot x.\theta.\epsilon_r,\ \ x.\theta.\epsilon_r,\ \ x.\theta.\lambda,\ \ x.\theta.\tau, \\ &\qquad x.\theta.\alpha,\ \ x.\theta.\beta,\ \ x.\theta.M)\,, \\ x^* &:= (x.U,\, x.C,\, \theta^*)\,, \\ A_G(x, \xi, r) &:= F(x^*, \xi).x\ . \end{aligned}$$

Note that in this case the new parameter $\theta.\epsilon_r$ is used to derive a temporary ϵ_n from the fraction r.

This concludes the formal definition of the RGNG algorithm.

2.2 RGNG-based neuron model

The RGNG algorithm as described above is used in the context of modeling a group of neurons as follows: The recursive depth of the RGNG is limited to one, which results in a two-layered structure with layers $L1$ and $L2$. The RGNG units of layer $L1$ correspond to the individual neurons of the modeled group. Each RGNG unit in $L1$ has a prototype that is a separate RGNG located in layer $L2$. This separate RGNG represents the dendritic tree of the corresponding neuron.

The RGNG in layer $L1$ is parameterized by θ_1. All RGNGs in layer $L2$ are parameterized by θ_2. Hence, the number of units (*neurons*) in $L1$ is set by $\theta_1.M$. The number of units (*prototypical input patterns in one dendritic tree*) per $L1$ unit is set by $\theta_2.M$. Therefore, the input space representation learned by the group of neurons as a whole utilizes $\theta_1.M \times \theta_2.M$ prototypical input patterns located in layer $L2$.

Inputs $\xi \in \mathbb{R}^n$ to the neuron model are processed through the input function F of the RGNG in $L1$. The resulting call graph is depicted in figure 1. Given

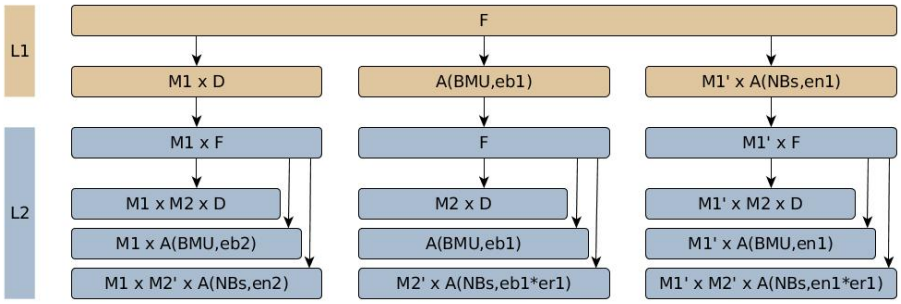


Fig. 1. Call graph for a single call to the input function F of an RGNG with two layers $(L1, L2)$. Executing F in layer $L1$ results in $\theta_1 . M$ (M1) calls to function D, a single call to function A with learning rate $\theta_1 . \epsilon_b$ (eb1), and $O(\theta_1 . M - 1)$ (M1') calls to function A with learning rate $\theta_1 . \epsilon_n$ (en1), where $O(\theta_1 . M - 1)$ is the potential size of the direct neighborhood of the corresponding BMU. The calls to D and A result in recursive calls to F on the next lower layer $L2$. Note that functions A temporarily change the learning rates for their calls to F. The recursion stops when a layer is reached where the prototypes are vectors; here L2. Calls to function I while the RGNGs are still growing are not shown since their computational cost is constant amortized. Figure adapted from [6].

that functions D and A are linear in n, and function I is constant amortized the computational cost to process a single, n-dimensional input by a two layer RGNG is $O(n \cdot \theta_1 . M \cdot \theta_2 . M)$.

After an input ξ is processed the output of the modeled group of $K := \theta_1 . M$ neurons is given as an ensemble activity $\mathbf{a} := (a_0, \ldots, a_{K-1})$ using a softmax function:

$$a_i := \frac{e^{\hat{a}_i}}{\sum_{j=0}^{K-1} e^{\hat{a}_j}},$$

with

$$\hat{a}_i := \gamma \left(1 - \frac{\|\mathbf{s_1^i} - \xi\|_2}{\|\mathbf{s_2^i} - \xi\|_2} \right), \quad i = 0, \ldots, K-1,$$

and $\mathbf{s_1^i}$, $\mathbf{s_2^i}$ being the best and second best matching prototypical input patterns found in the dendritic tree of neuron i, which are identified during the computation of function F of the corresponding $L2$ RGNG. The factor γ is used to control the degree with which the softmax function emphasizes the largest elements of the ensemble activity $\mathbf{a}$. From a neurobiological perspective the parameter γ can be interpreted as the strength of local inhibition that the neurons with the highest activations exert on the other neurons of the group.

The RGNG-based neuron group model was used successfully to describe the response behavior of typical entorhinal grid cells [6] as well as entorhinal cells that show grid like activity in response to saccadic eye movements [9, 14]. Furthermore, the model was analyzed regarding its resilience to noise [10, 13], and its

ability to perform pattern separation [11]. Lastly, it was integrated into a model of cortical column function as part of an autoassociative memory cell [12].

The aforementioned use cases of the RGNG-based neuron model focused on describing the response behavior of single neuron groups. Building on our latest work in modeling a cortical column [12] we aim to model large networks of cortical columns in our future research, i.e., to model hundreds or thousands of neuron groups. To this end it seems prudent to attempt to optimize the RGNG-based model with respect to its complexity, robustness, and computational requirements.

3 RGNG Approximation

The way the RGNG-based neuron group model forms a distributed, prototype-based representation of the group's input space is the key aspect of this neuron model. Two main processes are involved in the formation of this representation. First, each neuron has to learn a representation of the entire input space, i.e., it has to distribute the prototypical input patterns it learns across the input space rather than specialize in any one local region. Second, the representations of individual neurons have to be aligned in such a way that they are pairwise distinct and enable a disambiguation of inputs via the ensemble activity of the neuron group. In the RGNG-based model these two processes depend both on the dynamics encoded in input function F and are controlled by the learning rates $\theta_1 \cdot \epsilon_b / \cdot \epsilon_n$ and $\theta_2 \cdot \epsilon_b / \cdot \epsilon_n$ on layers $L1$ (group level alignment) and $L2$ (per neuron learning), respectively. Although the underlying growing neural gas algorithm is relatively robust regarding the choice of learning rates, it can still be difficult to identify a set of learning rates that ensure both the distribution of per-neuron prototypes across the input space and the alignment of the resulting representations such that the overall group-level representation of the input space settles into a stable configuration.

Another important aspect of the RGNG-based model is its computational complexity. For every input the algorithm has to calculate the distance of that input to each prototypical input pattern of every neuron. Since the distance calculation itself is not computationally demanding the RGNG algorithm becomes effectively I/O bound on typical, modern computer systems. When scaling a simulation from individual neuron groups to networks of hundreds or thousands of groups, i.e., exceeding the cache capacity of the system in use the I/O boundedness of the algorithm becomes the main limiting factor of that simulation regarding computation time.

To address both of these issues we present a novel variation of the growing neural gas algorithm, the *differential growing neural gas* (DGNG), and propose to use it as an approximation of the RGNG algorithm in the context of neuron group modeling. The main idea of the DGNG algorithm is to partition the input space into N regions, where N corresponds to the number of prototypical input patterns that are learned by each neuron. This partition is learned by a top layer growing neural gas with N units representing each input space region by a

single prototype vector located at the center of that region. Each region is then partitioned by separate sub-DGNGs into K sub-regions, where K corresponds to the number of neurons in the modeled neuron group. The units of these sub-DGNGs contain prototype vectors that represent a position relative to the center prototype of the corresponding region, i.e., the prototype vectors encode the respective difference between input and center prototype for a given input space sub-region.

Compared to the RGNG-based model the correspondence between $L1$ units and the neurons of the modeled neuron group has been removed in the DGNG. Instead, the representation of each neuron is now distributed among the different sub-DGNGs of the top layer DGNG units. More precisely, the i-th unit of sub-DGNG j contains the j-th prototype of neuron i.

Whereas the distribution and alignment of prototypes in the RGNG algorithm depend on a suitable choice of four different learning rates, the distribution and alignment of prototypes in the DGNG algorithm is directly enforced by its structure, thereby reducing the dependence on suitable learning rates for a particular problem instance. The partitioning of the input space in layer $L1$ ensures that the coarse input space representations of all neurons always cover the entire input space, while the competition in the $L2$ sub-DGNGs ensures that the representations learned by each neuron are pairwise distinct. In addition, the partitioning of the input space allows to reduce the number of distance calculations per input significantly (see section 3.2).

Analogous to section 2, the next section will introduce the DGNG algorithm independent of its use for modeling a group of neurons. Subsequently, section 3.2 will describe in more detail how the DGNG is used to describe the response behavior of a group of neurons.

3.1 DGNG Formal Description

Like an RGNG a DGNG g can be described by a tuple:

$$g := (U, C, \theta) \in G,$$

with a set U of units, a set C of edges, and a set θ of parameters. Each unit u is described by a tuple:

$$u := (w, e, s) \in U, \quad w \in \mathbb{R}^n, \;\; e \in \mathbb{R}, \;\; s \in G \cup \varnothing,$$

with the *prototype* w, the *accumulated error* e, and the *sub-DGNG* s. Note that in contrast to the RGNG the prototype w of an DGNG unit is always a n-dimensional vector, and the recursive structure is explicitly given by the sub-DGNG s, which is empty at the lowest level.
Each edge c is described by a tuple:

$$c := (V, t) \in C, \quad V \subseteq U \;\wedge\; |V| = 2, \;\; t \in \mathbb{N},$$

with the units $v \in V$ connected by the edge and the *age* t of the edge. The *direct neighborhood* E_u of a unit $u \in U$ is defined as:

$$E_u := \{k | \exists\, (V,t) \in C,\ \ V = \{u,k\}\,,\ \ t \in \mathbb{N}\}\,.$$

The set θ of parameters consists of:

$$\theta := \{\epsilon_b, \epsilon_n, \lambda, \tau, \alpha, \beta, M\}\,.$$

In contrast to the RGNG the behavior of a DGNG can be described by a single (recursive) input function F that feeds an input vector ξ into the DGNG g and returns the modified DGNG as well as the prototype vector $w^*$of the best matching unit :

$$F(g,\xi) : G \times \mathbb{R}^n \to G \times \mathbb{R}^n.$$

Note the subtle difference that this input function does not return the minimum distance to the input but rather the best matching prototype vector.

$\boldsymbol{F(g,\xi)}$*:* The input function F processes an input $\xi \in \mathbb{R}^n$ to the DGNG g as follows:

- If $g = \varnothing$ return g and a zero vector $\mathbf{0}$ as best matching prototype,
  else:
- Find the two units s_1 and s_2 with the smallest distance to the input ξ:

$$s_1 := \arg\min_{u \in g.U} \|u.w - \xi\|_p,$$
$$s_2 := \arg\min_{u \in g.U \setminus \{s_1\}} \|u.w - \xi\|_p.$$

- Increment the age of all edges connected to s_1:

$$\Delta c.t = 1, \quad c \in g.C \ \wedge\ s_1 \in c.V\ .$$

- If no edge between s_1 and s_2 exists, create one:

$$g.C \Leftarrow g.C \cup \{(\{s_1, s_2\}, 0)\}\,.$$

- Reset the age of the edge between s_1 and s_2 to zero:

$$c.t \Leftarrow 0, \quad c \in g.C \ \wedge\ s_1, s_2 \in c.V\ .$$

- Add the squared distance between ξ and the prototype of s_1 to the accumulated error of s_1:

$$\Delta s_1.e = \|s_1.w - \xi\|_p^2.$$

- Adapt the prototype of s_1 and all prototypes of its direct neighbors:

$$\Delta s_1.w = (\xi - (s_1.w + F(s_1.s, \xi - s_1.w).w^*))\, g.\theta.\epsilon_b,$$
$$\Delta s_n.w = (\xi - s_n.w)\, g.\theta.\epsilon_n.$$

Note that in this step a recursion takes place. The sub-DGNG s of the best matching unit s_1 is fed with the difference between the input ξ and the prototype w of s_1, and the returned best matching prototype w^* of the sub-DGNG s is used to augment $s_1.w$ in the current adaptation step. Thus, the sub-DGNG s can be understood as learning the details or different variations of the more coarse representation stored in $s_1.w$. The adaptation of the neighboring units is performed without recursion to reduce computational cost.

– Next, remove all edges with an age above a given threshold τ and remove all units that no longer have any edges connected to them:

$$g.C \Leftarrow g.C \setminus \{c | c \in g.C \wedge c.t > g.\theta.\tau\}\,,$$
$$g.U \Leftarrow g.U \setminus \{u | u \in g.U \wedge E_u = \emptyset\}\,.$$

– If an integer-multiple of $g.\theta.\lambda$ inputs was presented to the RGNG g and $|g.U| < g.\theta.M$, add a new unit u. The new unit is inserted "between" the unit j with the largest accumulated error and the unit k with the largest accumulated error among the direct neighbors of j. Thus, the prototype $u.w$ of the new unit is initialized as:

$$u.w := \frac{j.w + k.w}{2},\ j = \arg\max_{l \in g.U}(l.e)\,,$$
$$k = \arg\max_{l \in E_j}(l.e)\,.$$

The existing edge between units j and k is removed and edges between units j and u as well as units u and k are added:

$$g.C \Leftarrow g.C \setminus \{c | c \in g.C \wedge j, k \in c.V\}\,,$$
$$g.C \Leftarrow g.C \cup \{(\{j, u\}, 0), (\{u, k\}, 0)\}\,.$$

The accumulated errors of units j and k are decreased and the accumulated error $u.e$ of the new unit is set to the decreased accumulated error of unit j:

$$\Delta j.e = -g.\theta.\alpha \cdot j.e, \quad \Delta k.e = -g.\theta.\alpha \cdot k.e,$$
$$u.e := j.e\,.$$

– Finally, decrease the accumulated error of all units:

$$\Delta u.e = -g.\theta.\beta \cdot u.e, \quad \forall u \in g.U\,.$$

The function F returns the tuple (g, w^*) containing the now updated DGNG g and the prototype w^* of the best matching unit s_1 w.r.t. input ξ.

This concludes the formal definition of the DGNG algorithm.

3.2 DGNG-based Neuron Model

When modeling a group of neurons with the DGNG algorithm the recursive depth of the DGNG is limited to one like it is the case when using an RGNG.

However, the resulting two-layered structure is interpreted differently. There exists no longer a direct correspondence between certain DGNG units and neurons of the modeled neuron group. Instead, the prototypical input patterns learned by an individual neuron are distributed across the sub-DGNGs of all DGNG units of layer $L1$ and are composed as summation of the respective $L1$ unit's prototype vector and one prototype vector of the sub-DGNG's $L2$ units. Hence, the number of DGNG units in $L1$ corresponds to the number of prototypical input patterns learned by one neuron in it's dendritic tree, and the number of DGNG units in each sub-DGNG, i.e., in layer $L2$ corresponds to the number of neurons in the neuron group.

More specifically, given a group of K neurons that each learn N prototypical input patterns the set of prototypes $P^i := \left\{p_0^i, \ldots, p_{N-1}^i\right\}$ of neuron i within a DGNG g is defined as:

$$p_j^i := u_j.w + u_j.s.v_i, \quad u_j \in g.U, \;\; v_i \in u_j.s.U, \;\; j = 0, \ldots, N-1,$$

with $g.\theta.M = N$ and $u.s.\theta.M = K$, $\forall u \in g.U$. The computational complexity of processing an input $\xi \in \mathbb{R}^n$ with the DGNG-based model is significantly reduced compared to using an RGNG. Instead of requiring $O\left(n \cdot K \cdot N\right)$ operations the DGNG-based solution requires only $O\left(n\left(K + N\right)\right)$ operations.

The ensemble activity $\mathbf{a} := (a_0, \ldots, a_{K-1})$ of a DGNG-based neuron model is based on the sub-DGNGs of the best and second best matching DGNG units s_1 and s_2 in $L1$ and uses, like the RGNG-based model, a softmax function:

$$a_i := \frac{e^{\hat{a}_i}}{\sum_{j=0}^{K-1} e^{\hat{a}_j}},$$

with

$$\hat{a}_i := \gamma \left(1 - \frac{\|s_1.s.u_i.w - (\xi - s_1.w)\|_2}{\|s_2.s.u_i.w - (\xi - s_2.w)\|_2}\right), \quad i = 0, \ldots, K-1.$$

As with the RGNG-based model the factor γ is used to control the degree with which the softmax function emphasizes the largest elements of the ensemble activity $\mathbf{a}$.

4 Results

In order to perform a first characterization of a DGNG-based neuron group model we set up a model with 100 neurons, each of which had the capacity of learning 16 prototypical input patterns. We trained the model with inputs from the well-known MNIST dataset [16], which consists of 60000 grayscale images of handwritten digits with a resolution of 28×28 pixels. As parameters we used: $\theta.\epsilon_b := 0.01$, $\theta.\epsilon_b := 0.0001$, $\theta.\lambda := 1000$, $\theta.\tau := 300$, $\theta.\alpha := 0.5$, $\theta.\beta := 0.0005$, $\theta.M := \{16, 100\}$. We presented the MNIST training dataset repeatedly to the model until 10 million inputs were reached in total. After training the response of the modeled neuron group to the MNIST test dataset (10000 handwritten digits that were not in the training dataset) was stored as a set of

(a) (b) (c)

Fig. 2. Examples of learned prototypical input patterns. (**a**) Prototypes learned by the DGNG in $L1$ that partition the input space. (**b**) Relative prototypes of a single modeled neuron learned by different $L2$ DGNGs in the respective $L1$ input space regions. (**c**) Summation of the prototypes shown in **a** and **b** resulting in the actual patterns that are compared with the particular inputs.

ensemble activity vectors that were used in the following analysis. The ensemble activity vectors were sampled multiple times for varying values of output parameter γ (section 3.2).

The 16 prototypical input patterns learned by the DGNG in $L1$ that partition the input space are shown in figure 2a. As expected, the prototypes average over large regions of input space and thus show only vague patterns of typical, handwritten digits. Figure 2b shows the *relative* prototypes of a single modeled neuron. Each of these prototypes were learned by a different $L2$ DGNG associated with the corresponding $L1$ input space region and prototype. These relative $L2$ prototypes learn the difference between the average $L1$ prototype of their associated input space region and the more local input space region they represent. Combined, the average $L1$ prototype and the relative $L2$ prototype result in the patterns shown in figure 2c, which are those that are effectively used to determine the particular distance to a given input pattern and derive the activity of the particular neuron in response to that input. Figure 2c also illustrates that neurons in this neuron model do not specialize in a single type of input pattern, e.g., the digit 0, but respond to a variety of different input patterns such that a disambiguation of different input patterns has to happen at the neuron group level on the basis of the group's ensemble activity.

In the outlined DGNG-based neuron group model the group's ensemble activity in response to an input is a vector $\mathbf{a} \in \mathbb{R}^K$ with K being the number of neurons in the modeled group. The output parameter γ (section 3.2) controls the sparsity of this activity vector $\mathbf{a}$, which in turn determines the specificity of the neuron group's response to a given input. To analyze the effect the parameter γ has on the sparsity of $\mathbf{a}$ we calculated the distribution of Gini Indices of the ensemble activity vectors that were returned in response to the MNIST test dataset. The resulting distribution as a function of γ is shown in figure 3. The Gini Index can be used as a measure for the sparsity of a vector [5]. As an approximation, the value of the Gini Index can be intuitively understood as the

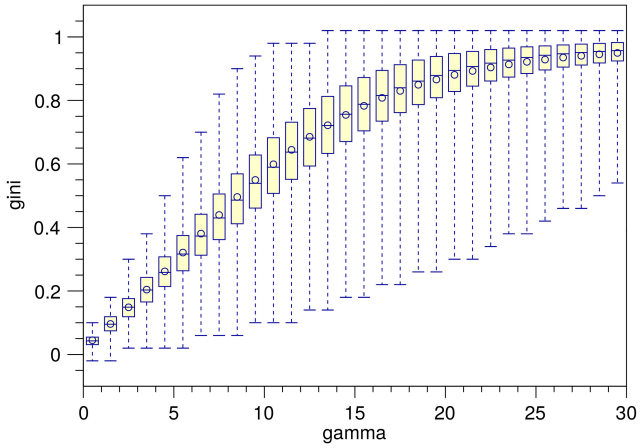


Fig. 3. Distribution of Gini Indices of the ensemble activity vectors of the modeled neuron group in response to the MNIST test dataset as a function of output parameter *gamma* (section 3.2). Whiskers indicate upper and lower quartiles, circles and center horizontal lines indicate mean and median of the distributions.

fraction of entries in a given vector that have low values (compared to the other entries), i.e., a Gini Index of 0 corresponds to a vector that has similar values in all of its entries whereas a Gini Index of 1 corresponds to a vector that has one or only a few entries with values significantly higher than all other entries of the vector.

The distribution shown in figure 3 illustrates that with increasing values of γ the mean sparseness of the ensemble activity vectors steadily increases as well. Considering the interpretation of γ as the degree of local inhibition in the proposed neuron model it becomes evident how important this local inhibition is to the formation of a sparse ensemble code that represents a given input with a sufficient degree of specificity. The ability to perform such a pattern separation was already a key characteristic of the RGNG-based neuron model [11]. To investigate the DGNG-based model in that regard we compared the pairwise cosine similarities of the ensemble activity vectors that were generated in response to the MNIST test dataset. Since the dataset provides labels for the ten different digit classes it was possible to compare the cosine similarities for intra- and inter-class inputs separately. Given that intra-class samples are likely to be more similar to each other than inter-class samples this distinction of cases allows to study the pattern separation abilities in more detail. In general it is expected that the group of neurons is able to represent individual inputs distinctively in both cases, although the task in the intra-class case is conceivably harder.

Figure 4 shows the resulting distributions of cosine similarities. In both cases the ensemble activity vectors for values of γ below 5 are very similar to each

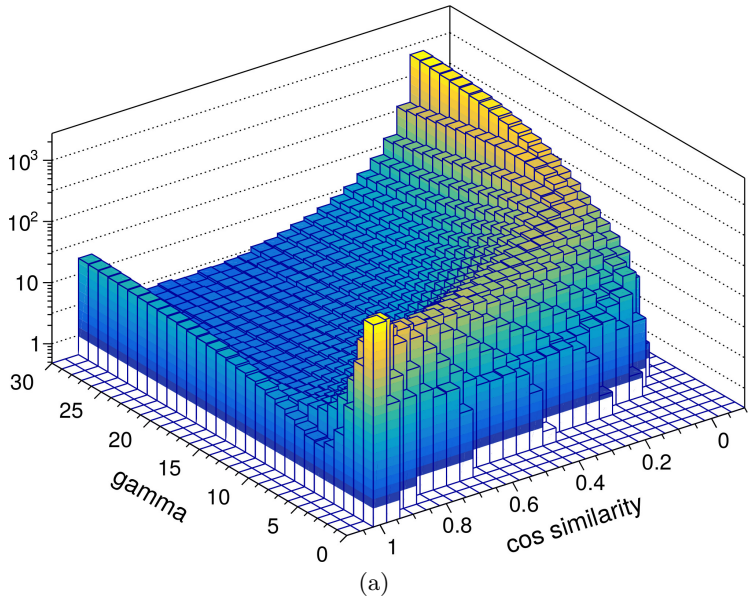
other and do not allow to distinguish the neuron group's responses to different inputs very well – again emphasizing the importance of local inhibition in this neuron model. However, with increasing values of γ (≥ 5), i.e., increasing local inhibition, the ensemble activity rapidly becomes very specific. Interestingly, the increase in specificity is not monotonic. The distributions for both intra-class and inter-class samples exhibit a minimum at $\gamma \approx 7$ and $\gamma \approx 5$ to 15, respectively. For these values of γ the response of the neuron group to a given input ξ is such that very few to none of the other test inputs have a cosine similarity close to one with respect to ξ while the spectrum of occurring, lower cosine similarity values is wide. In contrast, for values of $\gamma > 15$ the distinction between similar or dissimilar inputs becomes much more pronounced. The vast majority of other test inputs (note the logarithmic scale) exhibit cosine similarity values close to zero in that case, i.e., the corresponding ensemble activity vectors become almost orthogonal to each other.

From a neurobiological perspective the putative ability to shape the characteristic of the neuron group's input space representation just by the degree of local inhibition alone is a fascinating possibility. It would allow a group of neurons to dynamically switch between fine grained representations that enable the differentiation of very similar inputs and coarse grained, clear-cut representations suitable for fast classification.

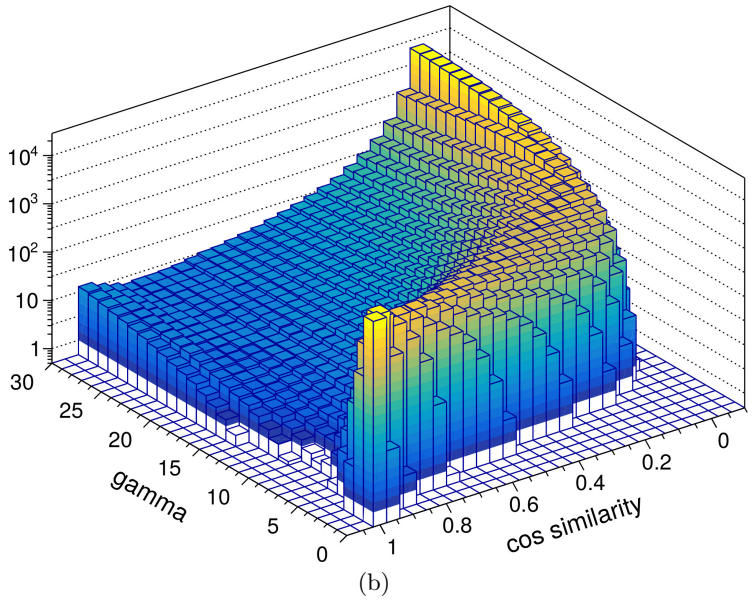
5 Conclusion

In this paper we presented a novel variant of the growing neural gas algorithm: the *differential growing neural gas* (DGNG). We designed this new algorithm as a drop-in replacement of the recursive growing neural gas (RGNG) in the context of modeling the response behavior of neuron groups. Compared to the old RGNG approach the new DGNG algorithm is more robust regarding the formation of the group's input space representation, is structurally less complex, and is computationally more efficient. In addition, the results of our first analysis of a DGNG-based neuron group model indicate that the model is able to retain important characteristics of earlier RGNG-based models.

Our future research will focus on investigating the characteristics of the DGNG algorithm further especially regarding its temporal stability, its capacity, and its use in our cortical column model.



(a)



(b)

Fig. 4. Histograms of pairwise cosine similarities between ensemble activity vectors of the modeled neuron group in response to the MNIST test dataset. Figure (**a**) shows cosine similarities for intra-class samples, while figure (**b**) shows the results for inter-class samples. In both cases, increasing values of γ lead to stronger pattern separation. Note the logarithmic scale of the z-axis.

References

1. Buxhoeveden, D.P., Casanova, M.F.: The minicolumn hypothesis in neuroscience. Brain **125**(5), 935–951 (2002)
2. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in Neural Information Processing Systems 7. pp. 625–632. MIT Press (1995)
3. Fyhn, M., Molden, S., Witter, M.P., Moser, E.I., Moser, M.B.: Spatial representation in the entorhinal cortex. Science **305**(5688), 1258–1264 (2004)
4. Hafting, T., Fyhn, M., Molden, S., Moser, M.B., Moser, E.I.: Microstructure of a spatial map in the entorhinal cortex. Nature **436**(7052), 801–806 (Aug 2005)
5. Hurley, N.P., Rickard, S.T.: Comparing measures of sparsity. CoRR (2008)
6. Kerdels, J.: A Computational Model of Grid Cells based on a Recursive Growing Neural Gas. Ph.D. thesis, FernUniversität in Hagen (2016)
7. Kerdels, J., Peters, G.: Analysis of high-dimensional data using local input space histograms. Neurocomputing **169**, 272 – 280 (2015)
8. Kerdels, J., Peters, G.: A new view on grid cells beyond the cognitive map hypothesis. In: 8th Conference on Artificial General Intelligence (AGI 2015) (July 2015)
9. Kerdels, J., Peters, G.: Modelling the grid-like encoding of visual space in primates. In: Proceedings of the 8th International Joint Conference on Computational Intelligence, IJCCI 2016, Volume 3: NCTA, Porto, Portugal, November 9-11, 2016. pp. 42–49 (2016)
10. Kerdels, J., Peters, G.: Noise resilience of an rgng-based grid cell model. In: Proceedings of the 8th International Joint Conference on Computational Intelligence, IJCCI 2016, Volume 3: NCTA, Porto, Portugal, November 9-11, 2016. pp. 33–41 (2016)
11. Kerdels, J., Peters, G.: Entorhinal grid cells may facilitate pattern separation in the hippocampus. In: Proceedings of the 9th International Joint Conference on Computational Intelligence, IJCCI 2017, Funchal, Madeira, Portugal, November 1-3, 2017. pp. 141–148 (2017)
12. Kerdels, J., Peters, G.: A grid cell inspired model of cortical column function. In: Proceedings of the 10th International Joint Conference on Computational Intelligence. pp. 204–210. INSTICC, SciTePress (2018)
13. Kerdels, J., Peters, G.: A Noise Compensation Mechanism for an RGNG-Based Grid Cell Model, pp. 263–276. Springer International Publishing (2019)
14. Kerdels, J., Peters, G.: A Possible Encoding of 3D Visual Space in Primates, pp. 277–295. Springer International Publishing (2019)
15. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**(1), 59–69 (1982)
16. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (Nov 1998)
17. Martinetz, T.M., Schulten, K.: Topology representing networks. Neural Networks **7**, 507–522 (1994)
18. Mountcastle, V.B.: The columnar organization of the neocortex. Brain **120**(4), 701–722 (1997)
19. Mountcastle, V.B.: An organizing principle for cerebral function: The unit model and the distributed system. In: Edelman, G.M., Mountcastle, V.V. (eds.) The Mindful Brain, pp. 7–50. MIT Press, Cambridge, MA (1978)