An energy-based convolutional SOM model with self-adaptation capabilities

Alexander Gepperth[1] and Ayanava Sarkar[1]

University of Applied Sciences Fulda, Leipzigerstr.123, 36037 Fulda, Germany
alexander.gepperth@cs.hs-fulda.de
http://www.hs-fulda.de

Abstract. We present a new self-organized neural model that we term ReST (**Re**silient **S**elf-organizing **T**issue), ReST can be run as a convolutional neural network (CNN), possesses a C^∞ energy function as well as a probabilistic interpretation of neural activities, which arises from the constraint of log-normal activity distribution over time that is enforced during learning. We discuss the advantages of a C^∞ energy function and present experiments demonstrating the self-organization and self-adaptation capabilities of ReST. In addition, we provide a performance benchmark for the publicly available TensorFlow-implementation.

Keywords: SOM · convolutional neural networks · self-adaption

1 Introduction

This article is in the context of self-organized map (SOM) models that have a continuous energy function. The lack of such an energy function for the original SOM model [7] has been the subject of many articles [2,11]. As it was shown that the original SOM learning rule cannot be derived from a continuous energy function [2], several proposals were made to remedy this problem [6,4]. In general, one may cite the following advantages of energy-based SOM models:

- **Estimation of learning success and parameter selection** A big issue for SOMs is to know whether the model has converged to a "desirable" state. For problems that do not allow a visual quality inspection, there is no universal criterion to determine optimal values for the model parameters (final neighbourhood radius, final learning rate etc.), whereas an energy function provides a simple quality measure.
- **Proof of stability** If a continuous energy function exists and is bounded from below, this automatically guarantees the eventual convergence of SOM learning.
- **Use of advanced stochastic gradient descent methods** With a continuous energy function, many widely-used methods for performing stochastic gradient descent (SGD) in the domain of deep learning can be transferred to SOM learning.
- **Outlier detection** A sudden increase of energy (which is supposed to be minimized by learning) is a strong indication for a change in data statistics and can thus be used for outlier or concept drift detection. The latter property is especially relevant for our own ongoing work on incremental learning methods[3].

1.1 Related work on energy-based SOM models

There has been a huge amount of primarily mathematical literature about It was shown conclusively in [2] that the original Kohonen learning rule cannot be exactly derived from the minimization of *any* error function. In the same article, it is mentioned that the Kohonen learning rule follows instead from the individual minimization of per-neuron energy functions [11], but these functions are very complex, non-unique and do not lend themselves to a simple interpretation (e.g., minimization of a distortion measure or similar). Another approach was proposed by Kohonen[7] and taken further by Heskes[6]: instead of finding error functions whose minimization would lead to the Kohonen learning rule, these authors attempted to very slightly modify the Kohonen rule itself. Obviously, the modification should in no way impair the self-organization capabilities of the model while allowing an intuitive interpretation through a (preferably simple) energy function. An modification satisfying these requirements was proposed in [6,5], offering a continuous energy function for discrete as well as continuous data distributions. While this was an important theoretical result, there was no real follow-up in terms of applications in data visualization and/or clustering.It may be supposed that this lack of interest was due to the added computational complexity (an additional convolution needs to be calculated), as well as the problems that convolutions encounter at boundaries. Similar SOM variants having an energy function were proposed in [4] but they suffer from the same "convolution problem".

2 Methods and data

In all experiments, we use the ReST model as described in Sec. 2.1, together with the MNIST dataset described in Sec. 2.2.

2.1 The ReST model

We assume a dataset (or a mini-batch) of input vectors $\boldsymbol{x}_n \in \mathbb{R}^k$ and a two-dimensional set of $K \times K$ neurons with non-negative activities $a_i \geq 0, i = 1 \ldots, K^2$. It is convenient to express activities computed for an input $\boldsymbol{x}_n$ as a one-dimensional vector $\boldsymbol{a}_n \in \mathbb{R}^{K^2}$. A neuron with (linear) index i and coordinates $x_i,\ y_i$ has an associated prototype $\boldsymbol{p}_i \in \mathbb{R}^k, i = 1, \ldots, K^2$, as well as an $K \times K$ neighbourhood matrix that we write as a one-dimensional vector $\boldsymbol{g}_i \in \mathbb{R}^{K^2}$ in analogy to the vector of activities. Differing from the SOM model, each neuron possesses two internal variables o_i and s_i that play a role in enforcing log-normal statistics for the activities $\boldsymbol{a}_n$ which are computed as:

$$d_{ni} = \sqrt{(\boldsymbol{p}_i - \boldsymbol{x}_n)^2} \tag{1}$$

$$\tilde{a}_{ni} = o_i - s_i d_{ni} \tag{2}$$

$$a_{ni} = \exp\left(\tilde{a}_{ni}\right). \tag{3}$$

The adaptation of the prototypes $\boldsymbol{p}_i$ is now achieved by minimizing the energy function

$$c_{ni} = \langle \boldsymbol{g}_i, \log \boldsymbol{a}_n \rangle = \langle \boldsymbol{g}_i, \tilde{\boldsymbol{a}}_n \rangle \tag{4}$$

$$\mathcal{E} = \frac{1}{N} \sum_n \langle \boldsymbol{c}_n, \boldsymbol{S}(\boldsymbol{c}_n) \rangle . \tag{5}$$

The first equation essentially represents a convolution operation as the per-neuron vectors $\boldsymbol{g}_i$ are (for self-organized models) represented by Gaussians centered on neuron i. Generally, one assumes such Gaussians to be periodic where they exceed the map boundaries (for neurons that are close to these boundaries). The logarithm and the vector-valued softmax function $\boldsymbol{S}(\boldsymbol{v})$ in eqn.(4) are applied in a component-wise fashion as

$$e_i = \exp(\beta v_i) \tag{6}$$

$$S(\boldsymbol{v})_i = \frac{e_i}{\sum_j e_j} \equiv S_i, \tag{7}$$

β being a parameter that controls the selectivity of the softmax: for higher β values, the output $S(\boldsymbol{v})$ will tend to be more strongly peaked, the maximal value closer to 1.0 and the rest to 0.0. For lower β values, this relationship is inversed. The minimization of the energy function is performed as a constrained optimization problem, the constraint being that the temporal distribution of activities $\boldsymbol{a}_n$ is log-normal with parameters μ and σ. This implies that $\log \boldsymbol{a}_n$ (with logarithm applied component-wise!) is normally distributed, with the empirical mean and standard deviation $\hat{\mu}$, $\hat{\sigma}$ coinciding with μ, σ:

$$\hat{\mu} \equiv \frac{1}{N} \sum_n \log a_{ni} = \frac{1}{N} \sum_n \tilde{a}_{ni} \stackrel{!}{=} \mu \tag{8}$$

$$\hat{\sigma} \equiv \sqrt{\frac{1}{N} \sum_n (\log a_i - \hat{\mu})^2} = \sqrt{\frac{1}{N} \sum_n (\tilde{a}_i - \hat{\mu})^2} \stackrel{!}{=} \sigma \tag{9}$$

From these requirements, the per-neuron parameters o_i and s_i can be determined unambiguously from the first two moments of the input-prototype distances

$$s_i = \sqrt{\frac{\sigma^2}{\overline{d_i^2} - \overline{d_i}^2}} \tag{10}$$

$$o_i = \mu + s_i \overline{d_i}, \tag{11}$$

which can be computed empirically over a dataset of N samples:

$$\overline{d_i} = \frac{1}{N} \sum_n d_{ni} \tag{12}$$

$$\overline{d_i^2} = \frac{1}{N} \sum_n d_{ni}^2$$

In a mini-batch setting, we instead take averages over the current mini-batch of N samples (the extreme case being fully online learning where $N = 1$). If we wish to

compute the averages $\overline{d_i}$ and $\overline{d_i^2}$ over periods longer than the mini-batch size N, we replace eqns.(12) by exponential smoothing of mini-batches averages:

$$\overline{d_i}(\nu) = (1 - \alpha_d N)\overline{d_i}(\nu - 1) + \alpha_d \sum_n d_{ni} \tag{13}$$

$$\overline{d_i^2}(\nu) = (1 - \alpha_d N)\overline{d_i^2}(\nu - 1) + \alpha_d \sum_n d_{ni}^2 \tag{14}$$

where variable ν expresses the number of the current mini-batch. We scale the adaptation rate $\alpha_d < 1$ with the mini-batch size N since a larger N implies that more samples are used per step in eqn.(13), and thus adaptation can proceed more quickly. Please note that by setting $\alpha_d = 0$ we can turn off the moving average mechanism. In this case only the current mini-batch is considered, as it is the case in eqn.(12).

ReST learning rule For performing gradient descent for the energy function of eqn.(4), we take its derivative w.r.t. to the k-th element of prototype i:

$$\frac{\partial E}{\partial p_{ik}} = \frac{\partial}{\partial p_{ik}} \frac{1}{N} \sum_{nj} c_{nj} S(\boldsymbol{c})_{nj} = \tag{15}$$

$$= \frac{1}{N} \sum_{nj} \left(S(\boldsymbol{c}_n)_j \frac{\partial c_{nj}}{\partial p_{ik}} + \beta S(\boldsymbol{c}_n)_i \left(\delta_{ij} - S(\boldsymbol{c}_n)_j\right) \right) \tag{16}$$

$$\approx \frac{1}{N} \sum_n \frac{\partial c_{n*}}{\partial p_{ik}} \tag{17}$$

where we have used the expression $\partial_j S_i = \beta S_i(\delta_{ij} - S_j)$ for the derivative of the softmax function. If we assume that the softmax function puts 1.0 at the position of the maximal value (whose index is expressed by $*$), and 0 everywhere else, we obtain the approximation result of eqn.(17) and arrive at the update rule

$$\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \frac{\epsilon s_i g_{*i}}{2N} \sum_n \frac{\boldsymbol{p}_i - \boldsymbol{x}_n}{||\boldsymbol{p}_i - \boldsymbol{x}_n||} \tag{18}$$

where we have one more time designed the index of the best-matching unit (BMU) by a star: $* = \arg\max_i c_i$. If we had omitted the square root in the definition of input-prototype distances in eqn. (1), we would have arrived at the equivalent rule

$$\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \frac{\epsilon s_i g_{*i}}{N} \sum_n \left(\boldsymbol{p}_i - \boldsymbol{x}_n\right) \tag{19}$$

which differs (for the online case of $N = 1$) from the energy-based SOM model proposed in [6] only by a factor of s_i for each neuron, an additional difference to [8] being that BMU is not determined from input-prototype distances but from the convolution $\boldsymbol{c}$ of activities with the neighbourhood matrix, see eqn. (4). We observe that the learning rules (18,19) scale each neuron's prototype adaptation by a factor which is, by eqn. (10), inversely proportional to the activity variance of that neuron. Thus, neurons whose

prototypes are either too unspecific or too generic (resulting in uniformly low or high activations with low variance) receive a competitive advantage. This mechanism is self-limiting: increased prototype adaptation usually increases the variance of a neuron's activities, thus eventually annulling the competitive advantage and leading to stable competitive learning dynamics.

Implementation of constrained optimization Minimizing the energy function (4) is performed by performing repeated gradient descent steps using learning rule (18) on the whole available training data set or mini-batch, each step followed by an explicit enforcement of the constraints by applying eqn. (10), this again being followed by an update of the averages using eqn.(12). For speeding up convergence, the neighbourhood matrix $\boldsymbol{g}_i$ of neuron i is modelled as a Gaussian whose standard deviation $S(\nu)$ is decayed exponentially over time, as it is usual with SOMs:

$$g_{ij} = \exp\left(-\frac{(x_j - x_i)^2 + (y_j - y_i)^2}{2S(\nu)^2}\right) \tag{20}$$

In contrast to normal SOM learning, we do not decay the ReST learning rate α over time, since this complicates advanced gradient descent strategies and introduces unnecessary parameters. Additionally, we impose an initial period without prototype adaptation where only neural statistics are adapted. This allows "adiabatic" prototype updates, causing only small corrections to the already converged o_i and s_i, which avoids potentially problematic feedback loops between the two adaptation processes. The training procedure, as well as all relevant parameters, is detailed in Alg. 1

Choice of ReST parameters The self-adaptation process is governed by the parameters μ and σ of the log-normal distribution that the activities a_i are required to obey, which raises the question of what their intrinsic significance could be, especially within the context of self-organizing maps and incremental learning. First of all, from the properties of log-normal distributions we know that the quantity e^{μ} represents both the geometric mean and at the same time the median of a log-normally distributed variable, so essentially we could just fix a median value M and compute $\mu = \log M$ from it. The median for this distribution is smaller but usually close to the arithmetic mean as well so we can also see M as a rough indicator for the arithmetic time average of a neuron's activity. The quantity e^{σ} is sometimes termed the geometric standard deviation and can be expressed as

$$\begin{aligned} e^{\sigma} &= \exp\left(\sqrt{\frac{1}{N}\sum_n \left(\log\frac{a_{ni}}{e^{\hat{\mu}}}\right)^2}\right) = \\ &= \sqrt[N]{\Pi_n \exp\left(\left(\log\frac{a_{ni}}{e^{\hat{\mu}}}\right)^2\right)} = E_n^g \sqrt{\exp\left(\left(\log\frac{a_{ni}}{e^{\hat{\mu}}}\right)^2\right)} \end{aligned} \tag{21}$$

and is thus related to the geometric mean of the expression $\sqrt{\exp\left(\left(\log\frac{a_{ni}}{e^{\hat{\mu}}}\right)^2\right)}$. This expresses the multiplicative spread of values around their empirical geometric mean

Algorithm: Constrained ReST optimization

Parameters :

- nr of iterations T
- mini-batch size N
- initial and final neigh. radius S_0, S_∞
- learning rate α
- self-adaptation rate α_d
- time parameters t_A, t_0 and t_∞
- target values σ, μ for self-adaptation

Result: trained prototypes $\boldsymbol{p}_i$
begin
  Initialize all prototypes $\boldsymbol{p}_i$ to small random values ;
  Initialize moving averages $\overline{d_i}(0) = 0$ and $\overline{d_i^2} = 0$;
  Initialize per-neuron parameters $s_i = 0.5$, $o_i = 0$;
  Compute decay time constant $\lambda = -\frac{\log(-S_\infty/S_0)}{t_\infty - t_0}$;
  for *mini-batch* $\nu < T$ **do**
    compute nb.radius $S(\nu)$ and learning rate $\alpha(\nu)$: **begin**
      if $\nu < t_A$ **then** $\alpha(\nu) = 0$, $S(\nu) = S_0$;
      else if $\nu < t_0$ **then** $\alpha(\nu) = \alpha$, $S(\nu) = S_0$;
      else if $\nu < t_\infty$ **then** $\alpha(\nu) = \alpha$, $S(\nu) = S_0 e^{-\lambda\nu}$;
      else $\alpha(\nu) = \alpha$, $S(\nu) = S_\infty$;
    end
    recompute nb. matr. $\boldsymbol{g}_i$ based on $S(\nu)$;
    select a random mini-batch $\boldsymbol{x}_n$, $0 < n < N$;
    update prototypes $\boldsymbol{p}_i$ according to eqn. (18) ;
    enforce constraint using eqn. (10) ;
    adapt averages $\overline{d_i}(\nu)$ and $\overline{d_i^2}(\nu)$ using eqn. (13) ;
  end
  return $\boldsymbol{p}_i$
end

Algorithm 1: Mini-batch based learning with the ReST model.

$e^{\hat{\mu}}$, regardless of the direction. Higher values of e^σ will push the activities further away from their geometric mean, forcing them to be more specific, either close to 0 or far away from it. We can thus think of σ as a parameter controlling the sparsity of neural responses, which previous studies on transfer functions for self-organized maps [10] found to be an important factor for performing classification based on SOM activities.

In order to guarantee identical functioning of the WTM mechanism for variable map sizes, the softmax function needs to be parameterized correctly, and more specifically as a function of the number of neurons in the SOM. We therefore need to set the parameter β such that qualitatively identical behavior ensues for any map size. We measure identical behavior by demanding that the the maximal response of the softmax function be ξ when given a vector $\boldsymbol{x} \in \mathbb{R}^n$ that consists of $n-1$ times value B and 1 time value

λB. Solving this for β gives us the expression

$$\beta = \frac{\ln(\xi^{-1} - 1) - \ln(n - 1)}{B(1 - \lambda)} \tag{22}$$

The softmax function is a very useful tool for obtaining a "hard" yet differentiable winner selection, in addition to allowing a steady transition between "hard" and "soft" winner selection. In some cases, problems can occur: first of all, sensible choices for B and λ may be hard to obtain because they depend on the learning dynamics. Furthermore, when $\beta > 700$, numerical issues arise due to the exponentials involved. Fortunately there is a simple rule-of-thumb solution for both problems that consists of applying a softmax function with "best guess" parameters *several times* in eqn. (4). This complicates the gradient, but as long as the final softmax function gives a sufficiently hard winner assignment, the learning rule (18) remains valid. Software frameworks like TensorFlow can compute the gradient symbolically, so even the exact gradient can be used regardless of how often softmax was applied. We found that a three-fold application was always sufficient to guarantee a unique winner selection.

The parameter S_0 is usually made to depend on the map size. A rule of thumb that always worked well is to choose it proportional to the diagonal of the quadratic $K \times K$ map, i.e., $S_0 = \frac{K}{4}$. In contrast, classification experiments always give best results the smaller S_∞ is, so this is always fixed at small values like $S_\infty = 0.01$. The values of t_0, t_A and t_∞ can be determined empirically be requiring that i) self-adaptation has occurred before t_A ii) the energy function has converged to a stable value before t_0 and iii) that the energy function is as low as possible while still satisfying all constraints at t_∞. Here, we see the value of an energy function as it can be used to determine convergence, so these parameters which for SOMs have to be obtained by visual inspection, can be determined by cross-validation. By a similar reasoning, a good value for the learning rate can be obtained, where smaller values are always acceptable but lead to increased training time. The mini-batch size is generally assumed to be $N = 1$ in this article. The self-adaptation rate, $\alpha_d N$, should be chosen such that the constraints are approximately upheld during prototype adaptation, meaning it will depend on the choice of α and is thus not a free parameter but can be indirectly obtained by cross-validation.

2.2 Data

We rely on the well-known MNIST benchmark [9] for handwritten digit recognition that is a standard problem in machine learning. For our purposes, it is ideal for testing our implementations as it allows a visual inspection of the learned prototypes, facilitating the detection of implementation errors through obviously corrupted prototypes. The MNIST dataset contains 60.000 training samples in 10 classes that are approximately equiprobable, as well as 10.000 samples in the test set.

3 Experiments

The ReST model used in all experiments is implemented in Python using TensorFlow 1.5 [1]. The gradients (18,15) are computed automatically by the software. Energy mini-

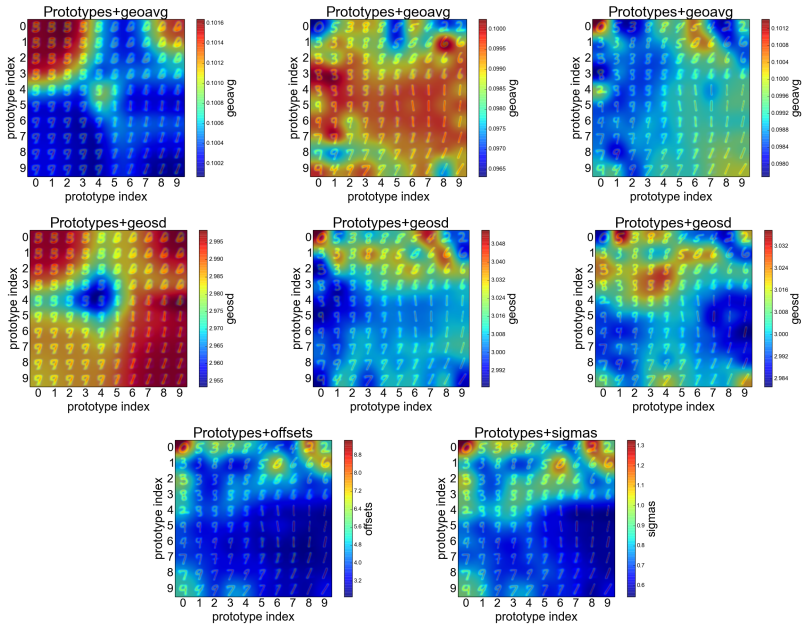

Fig. 1. Upper two rows: Different stages of ReST training on the MNIST dataset. Upper row, from left to right: ReST prototypes with long-term geometric activity averages superimposed on them for times $t = 7000, 12000, 24000$. Middle row, from left to right: ReST prototypes with long-term geometric standard deviation averages superimposed on them for times $t = 7000, 12000, 24000$. We observe that activity averages and deviations are strictly adhered to, as well as the SOM-like topological organization of prototypes. Lower row: Distribution of per-neuron parameters o_i and s_i after convergence of the ReST layer at iteration 24000.

mization is done by plain stochastic gradient descent, although more advanced optimizers minimize the ReST energy function equally well.

3.1 Self-organization and self-adaptation in the ReST model

In this section we will demonstrate that the ReST model, while differing from both the original SOM model [8] and the energy-based "Heskes model" [6], achieves the same basic type of prototype self-organization. At the same time, we will demonstrate the effectiveness of ReST's self-adaptation process as described in Sec. 2.1 and comment on its beneficial effects. To this end, we will conduct simulations with both datasets described in Sec. 2.2. ReST parameters are chosen as follows (in the terms of Sec. 2.1): $K = 10$, $T = 40000$, $t_A = 5000$, $t_0 = 10000$, $t_\infty = 30000$, $S_0 = K/4$, $S_\infty = 0.1$, $\alpha_d = 0.01$, $\alpha = 0.05$, $e^\sigma = 3$ and $e^\mu = 0.1$. After ReST convergence at t_∞, statistics is collected for 5000 iterations and subsequently evaluated. Histograms of all neural activities during these 5000 iterations are computed and compared to the theoretical log-normal distribution determined by μ and σ. From Fig. 1, it can be observed that self-organization proceeds exactly in the same manner as in a SOM, starting with a coarse

"global ordering" of prototypes followed by refinement as $S(\nu)$ is decreased, showing that ReST performs essentially the same function as a SOM, only with convergence in 2D guaranteed and a self-adaptation process that give a probabilistic interpretation to the computed activities. As can be seen in Fig. 2, the fit between theoretical and measured distribution is generally acceptable for all datasets, although of course a perfect fit is not to be expected. This is because we only fit the first two moments of the log activities to defined values. For a better fit, at least the third moment of the log activities should be controlled, which would however result in a more complex constrained optimization scheme. Fig. 1 shows this homogeneity is achieved by quite heterogeneous settings of the per-neuron parameters o_i and s_i, see eqn. (1).

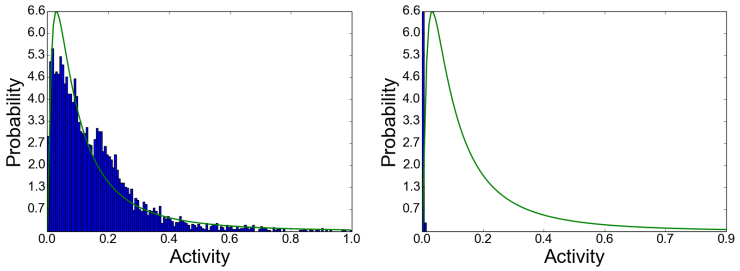

Fig. 2. Activity histograms for neuron $(4, 4)$ in a ReST layer trained on MNIST both for the case of disabled (left) and enabled (right) self-adaptation. The theoretical log-normal density is superimposed onto the histograms as a solid green line, showing a very good match.

3.2 Convolutional ReST experiments

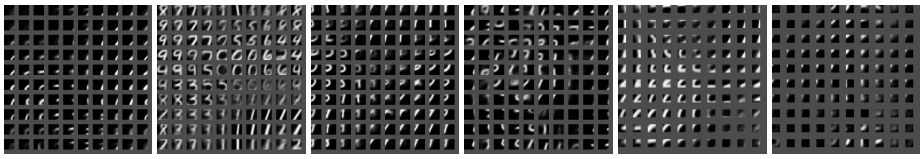

Fig. 3. Prototypes for convolutional/independent ReST architectures (left to right), defined by f_x^H, Δ_x^H, y, x: ind-14-7-0-0, ind-14-7-1-1, ind-14-7-2-2, conv-14-7, ind-7-3-3-3, ind-7-3-6-6.

As with CNNs, convolutional ReST layers have a great number of possible configurations for the filter sizes (f_x^H, f_y^H) and step sizes (Δ_x^H, Δ_y^H), of which we can test only a few. Experimental outcomes are the learned filters for each configuration as shown in Fig. 3, where we see that ReST performs both topological organization (as in a SOM) as well as feature extraction (as in a CNN layer).

3.3 Intuitive interpretation of the self-adaptation process

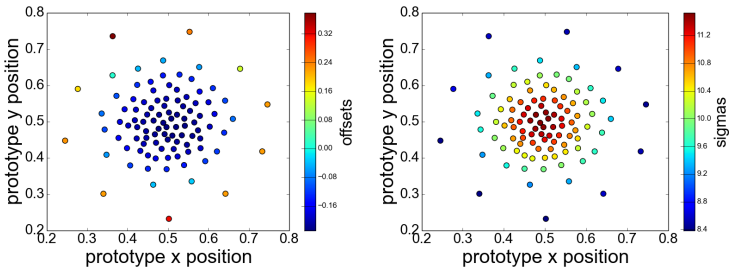

Fig. 4. Prototype positions overlaid in color with per-neuron parameters o_i (left) and s_i (right) when training a ReST layer on a 2D normal distribution.

To better understand what the self-adaptation mechanism in ReST actually does, we create a set of 10.000 two-dimensional data points $\boldsymbol{x}_i \in \mathbb{R}^2$ which are drawn from a normal distribution with mean $\boldsymbol{\mu} = (0.5, 0.5)^T$ and standard deviation $\Sigma = 0.15$. We subsequently train a non-convolutional ReST layer of size $K = 10$ using the parameters of Sec. 3.1. The final prototype positions and values of the per-neuron parameters s_i and o_i are shown in Fig. 4 and shows the following things:

- where data points are more dense(sparse), overall offsets o_i are lower(higher). This is intuitive since prototypes that react to less frequently occurring samples need to have a higher offset to maintain a constant average activity.
- where data points are more dense(sparse), selectivities s_i are higher(lower), meaning that a neuron will react less(more) strongly to nearby samples. This is intuitive as well, since a higher number of nearby samples would mean a near-constant activity, with low variance, if neurons could not become more selective in their reactions.

These results show that ReST neurons can adapt to the sample density in their Voronoi cell, a behavior that closely mimicks self-adaptation mechanisms in biological neurons.

3.4 Implementation and GPU speed-up

Unless otherwise stated, benchmark experiments always use the following parameters: Map size $H^H = W^H = 10$, $\Sigma_0 = 2$, $\epsilon_0 = 0.1$, $\epsilon_\infty = 0.0001$, $\Sigma_\infty = 0.01$, $T_{\text{conv}} = 3000$, $T_{\text{conv}} = 10000$. We compare the execution time per sample by feeding the SOM model 2000 randomly selected samples, either running it on CPU or GPU (NVIDIA GeForce 1080), and vary the map size $W^H = H^H \in \{10, 15, 20, 30, 50\}$ and the input batch size $N^I \in \{1, 5, 10, 20, 50, 100\}$ independently. The results of Fig. 5 show that, first of all, GPU acceleration is most effective at high batch sizes and amounts to a factor of roughly 10-20 w.r.t. CPU speed. Secondly, as expected, for high batch and map sizes the GPU is saturated, resulting in no more speed improvements from parallelization.

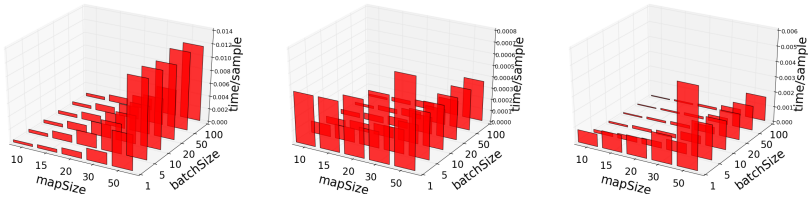

Fig. 5. ReST execution speed depending on batch size and map size, measured on: CPU without updating(left), GPU without updating (middle), GPU with updating (right).

And lastly, updating the SOM incurs a heavy speed penalty even on GPU, probably because of the convolution in eqn.(4).

4 Discussion and conclusion

We have presented a conceptually new and computationally feasible model of self-organized learning that mimicks biological self-adaptation processes as a part of its learning mechanism.

References

1. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 7
2. E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological cybernetics*, 67(1):47–55, 1992. 1, 2
3. A. Gepperth and C. Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, pages 1–11, 2016. 1
4. T. Graepel, M. Burger, and K. Obermayer. Self-organizing maps: Generalizations and new optimization techniques. *Neurocomputing*, 21(13):173 – 190, 1998. 1, 2
5. T. Heskes. Energy functions for self-organizing maps. In *Kohonen maps*, pages 303–315. Elsevier, 1999. 2
6. T. M. Heskes and B. Kappen. Error potentials for self-organization. In *Neural Networks, 1993., IEEE International Conference on*, pages 1219–1223. IEEE, 1993. 1, 2, 4, 8
7. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybernet.*, 43:59–69, 1982. 1, 2
8. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybernet.*, 43:59–69, 1982. 4, 8
9. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In S. Haykin and B. Kosko, editors, *Intelligent Signal Processing*, pages 306–351. IEEE Press. 7
10. M. Lefort, T. Hecht, and A. Gepperth. Using self-organizing maps for regression: the importance of the output function. In *European Symposium On Artificial Neural Networks (ESANN)*, 2015. 6
11. V. Tolat. An analysis of kohonen's self-organizing maps using a system of energy functions. *Biological Cybernetics*, 64(2):155–164, 1990. 1, 2