University of Groningen

Fusion of deep learning architectures, multilayer feedforward networks and learning vector quantizers for deep classification learning

Villmann, T.; Biehl, M.; Villmann, A.; Saralajew, S.

Published in:

12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)

DOI:

[10.1109/WSOM.2017.8020009](https://doi.org/10.1109/WSOM.2017.8020009)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version

Publisher's PDF, also known as Version of record

Publication date:

2017

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Villmann, T., Biehl, M., Villmann, A., & Saralajew, S. (2017). Fusion of deep learning architectures, multilayer feedforward networks and learning vector quantizers for deep classification learning. In *12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)* (pp. 1-8). IEEEExplore. <https://doi.org/10.1109/WSOM.2017.8020009>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Fusion of Deep Learning Architectures, Multilayer Feedforward Networks and Learning Vector Quantizers for Deep Classification Learning

T. Villmann*, M. Biehl[†], A. Villmann^{*‡}, and S. Saralajew[§]

*University of Applied Sciences Mittweida
Computational Intelligence Group
Mittweida, Germany

[†]University Groningen
Intelligent Systems Group
Groningen, The Netherlands

[‡]Berufliches Schulzentrum Döbeln-Mittweida
Mittweida, Germany

[§]Dr. Ing. h.c. F. Porsche AG
Electr./Electronics Eng. - Driver Assistance Platform/ Systems,
Weissach, Germany

Abstract—The advantage of prototype based learning vector quantizers are the intuitive and simple model adaptation as well as the easy interpretability of the prototypes as class representatives for the class distribution to be learned. Although they frequently yield competitive performance and show robust behavior nowadays powerful alternatives have increasing attraction. Particularly, deep architectures of multilayer networks achieve frequently very high accuracies and are, thanks to modern graphic processor units use for calculation, trainable in acceptable time.

In this conceptual paper we show, how we can combine both network architectures to benefit from their advantages. For this purpose, we consider learning vector quantizers in terms of feedforward network architectures and explain how it can be combined effectively with multilayer or single-layer feedforward network architectures. This approach includes deep and flat architectures as well as the popular extreme learning machines. For the resulting networks, the multi-/single-layer networks act as adaptive filters like in signal processing while the interpretability of the prototype-based learning vector quantizers is kept for the resulting filtered feature space. In this way a powerful combination of two successful architectures is obtained.

I. INTRODUCTION

The availability of powerful computers and, particularly, the use of graphic processing units (GPU) have led to a revival of multilayer perceptron (MLP) and convolutional neural network (CNN) learning for advanced data processing. Further, the tremendous increase of labeled data, e.g. images in social media, provide a huge amount of data for supervised learning.

Although the basic mathematical tools for supervised neural network training were proposed already in the seventies of the last century (see [1], [2] for MLP and [3] for CNN), the return of these networks into the focus of research and application

was stimulated by the consideration of deep architectures [4], [27]. Those deep networks consist of many hidden layers, each of them trained to be responsible for certain data features whereas the higher the layer the more complex the features are [4], [5]. In this sense, the lower layers can be seen as adaptive filters as it was already considered in [6] for MLPs.

The deep learning approaches make an essential use of the GPU implementations accelerating the computations needed for network learning enormously. Thus, the computational speedup results in learning times magnitudes lower than computations carried out on conventional CPUs [7].

A disadvantage of MLP or deep architectures is that the trained networks are essentially black-box systems, i.e. the functionality of each subnet/layer is difficult to interpret. This is in contrast to prototype based networks like learning vector quantization for classification (LVQ, [8], [9]), which are intuitively understandable due to their reference principle based on dissimilarity comparisons between data and representative prototypes [10], [11]. Further, in recent years, many improvements of the basic LVQ schemes were proposed which have led to a strong mathematical foundation based on cost functions (Generalized LVQ - GLVQ [43]) as well as to sophisticated variants covering many advanced learning problems like automatic dissimilarity adaptation and feature relevance learning, learning from imbalanced data or optimization of statistical measure instead of simple accuracy learning [12], [13].

Therefore the question arises, whether one can combine MLP/CNN/deep architecture with LVQ methods [14]. Doing so, the deep architectures serve as an adaptive filter for the LVQ-output-layer. The LVQ-prototypes act in the filter feature space as interpretable data models.

In this paper we propose strategies and network architec-

Figure 1. Illustration of a MLP network

tures, which combine MLP's or deep networks with LVQ models to fuse their advantages. In fact, we propose to replace the last MLP layer by a cost-based LVQ variant, which is in fact, replacing the L2- or cross entropy loss by misclassification costs according to the GLVQ classifier function.

II. MULTILAYER ARCHITECTURES AND EXTREME LEARNING MACHINES

A. Multilayer Perceptrons as Feedforward Networks

One of the most popular neural networks for supervised learning is the MLP [15]. The MLP consists of one input layer $\mathbf{I} \in \mathbb{R}^{N_I}$, several hidden layers $\mathbf{h}^k$, $k = 1, \dots, K$ with $\mathbf{h}^k \in \mathbb{R}^{N_k}$ and an output layer $\mathbf{O} \in \mathbb{R}^{N_O}$, see Fig. 1.

The input layer receives signals $\mathbf{x}_j \in \mathbb{R}^n$ with $n = N_I$ and the output delivers a corresponding vector $\mathbf{y}_j \in \mathbb{R}^m$ with $m = N_O$. Thereby, a signal $\mathbf{x}$ is fed to the first hidden layer according to

$$h_i^1 = g_1 \left(\sum_{j=1}^{N_I} W_{ij}^1 x_j + \beta_i^1 \right)$$

and then propagated through the hidden layers by means of

$$h_i^k = g_k \left(\sum_{j=1}^{N_{k-1}} W_{ij}^k h_j^{k-1} + \beta_i^k \right)$$

to generate the output

$$O_i = g_O \left(\sum_{j=1}^{N_K} W_{ij}^O h_j^K + \beta_i^O \right)$$

of the MLP. Here $\mathbf{W}^k$ and $\mathbf{W}^O$ are the weight matrices, β_k and β^O are the bias vectors whereas all the g_k and g_O are the activation functions usually to be chosen as non-linear sigmoid functions [16]. Frequently, all activation functions are equal to reduce the complexity of the network. A usual choice is

$$\phi(z, \theta) = \frac{1}{1 + \exp\left(\frac{z}{\theta}\right)} \quad (1)$$

for training or the function $\tanh(z \cdot \theta)$ [1]. Further we emphasize at this point that the layers compute their outputs

depending on the Euclidean inner products $\langle \mathbf{w}_i^k, \mathbf{h}^{k-1} \rangle_E = \sum_{j=1}^{N_{k-1}} W_{ij}^k h_j^{k-1}$ realizing the Hebbian response. Yet, in mathematical sense, other (differentiable) inner products could be used instead the most prominent of which are kernels [17], [18].

The supervised training of MLP is known as back-propagation minimizing the mean squared error between the output $\mathbf{y}_k = \mathbf{O}(\mathbf{x}_k)$ of the network for a sample $\mathbf{x}_k$ and the desired target output $\mathbf{t}_k$, i. e.

$$E_{MLP} = \sum_k (\mathbf{y}_k - \mathbf{t}_k)^2 \quad (2)$$

is optimized. Further, the bias vectors can be simulated by an additional component in the hidden layers and are, therefore, frequently omitted as we did in Fig. 1.

However, it turns out that an MLP with only one hidden layer $\mathbf{h} \in \mathbb{R}^{N_h}$ (i. e. $K = 1$ above) is sufficient to approximate an arbitrary function $f : \mathbb{R}^n \ni \mathbf{x} \rightarrow \mathbf{y} \in \mathbb{R}^m$, i. e. the MLP with only one hidden layer is an universal function approximator [19]. If all activation functions would be linear, the network would realize a linear mapping. Thus, the non-linearity of the activation functions is essential for the general approximation ability of the MLP. With this observation we have the reduced system

$$h_i = g_h \left(\sum_{k=1}^n W_{ik}^h x_k + \beta_i^h \right) \quad (3)$$

with sigmoid transfer function g_h and weight matrix $\mathbf{W}^h \in \mathbb{R}^{N_h \times n}$ describing the data processing from input $\mathbf{I}$ to the hidden layer followed by

$$y_j = g_O \left(\sum_{k=1}^{N_h} W_{jk}^O h_k + \beta_j^O \right)$$

to generate the network output. Furthermore, it turns out that the universal approximation property of the single-hidden-layer feedforward network (SLFN) is preserved if either the g_h -function or the g_O -function are non-linear [20], [21].

At this point we note that (3) can be rewritten as

$$h_i = g_h \left(\langle \mathbf{w}_i^h, \mathbf{x} \rangle_E + \beta_i^h \right) \quad (4)$$

with $\langle \mathbf{w}_i^h, \mathbf{x} \rangle_E$ being the standard Euclidean inner product between the i -th row vector $\mathbf{w}_i^h$ of $\mathbf{W}^h$ and $\mathbf{x}$. In this sense, we can associate each node h_i in the hidden layer with a weight vector $\mathbf{w}_i^h$. Replacing the inner product by an arbitrary non-linear kernel $\kappa_\Phi(\mathbf{w}_i^h, \mathbf{x})$ corresponds to a kernel mapping Φ to a reproducing kernel Hilbert space adopting the kernel trick of support vector machines [22], [17], [23].

Instead of minimizing the standard cost function E_{MLP} one can think about more sophisticated cost functions. If all entries of the target outputs $\mathbf{t}_k$ are non-negative alternatives would be (generalized) divergences $\mathcal{D}(\mathbf{y}_k, \mathbf{t}_k)$ or respective mutual information $\mathcal{M}(\mathbf{y}_k, \mathbf{t}_k)$ to be optimized by stochastic gradient descent learning (SGDL) [24], [25], [26].

Another loss function, which became popular during the last years is the cross entropy loss

$$E_{MLP-CE} = - \sum_k \sum_{i=1}^m (t_{k,i} \ln(p(y_{k,i} | \mathbf{x}_k))) \quad (5)$$

with the probabilistic output model $p_O(y_{k,i}|\mathbf{x}_k)$ and

$$L_k = \sum_{i=1}^m (t_{k,i} \ln(p_O(y_{k,i}|\mathbf{x}_k))) \quad (6)$$

as local cross entropy loss [27, p. 178ff]. The probability $p_O(y_{k,i}|\mathbf{x}_k)$ is usually approximated by the softmax function

$$p_O(y_{k,i}|\mathbf{x}_k) \approx s_{\max}(y_{k,i}) = \frac{\exp(y_{k,i})}{\sum_{j=1}^m \exp(y_{k,j})}$$

for each entry in $\mathbf{y}_k$ and g_O is the identity function. Thus the derivative of the local loss L_k becomes

$$\frac{\partial L_k}{\partial y_{k,j}} = (s_{\max}(y_{k,i}) - t_{k,j}) s_{\max}(y_{k,i}) (t_{k,j} - s_{\max}(y_{k,i})) \quad (7)$$

for easy calculation. It turns out that the cross entropy loss accelerates the convergence and is more appropriate than the standard squared error (2) [27, p. 178ff].

Another improvement, frequently applied in the context of deep learning as explained in the following, is the replacement of the sigmoid activation function (1) by rectified linear units (ReLU)

$$\rho(z) = \max\{0, z\} \quad (8)$$

simplifying the adaptation rules [28]. Although this function is not bounded and differentiable everywhere, most of the recent applications make use of ReLU's [5]. Both difficulties can be avoided using regularization techniques.

Last but not least, advanced gradient descent variants provide powerful alternatives to the simple SGDL. The respective algorithms frequently include higher order information like in RMSProp [29], AdaGrad [30], AdaDelta [31], Adam [32] for adaptation of the gradient step size, or are variance-based SGDL schemes (vSGDL) as proposed in [33]. These approaches have led to significant performance improvement also for MLP-learning.

B. Deep Architectures

Deep architectures differ from the SLFN by the large depth K whereas the dimensionalities N_k of the hidden layers are usually low. Moreover, starting with a large number of N_I of input layers, which are not necessarily required to be fully connected to all inputs and the possibilities of parallel hidden layers at the same level which also have not to be fully connected to the previous layer and the following layer, the deep architecture provides a great flexibility [34]. Moreover, it allows a separate unsupervised pre-training of parts of the network to make them responsible for certain predefined data properties [35], see Fig. 2.

In fact, the subnets can be seen itself as MLPs but have to be learned in an unsupervised manner. In this sense they can be taken as non-linear filters as it was already discussed in context of information theoretic learning [6], [24]. Further, the subnets can be of different structures. For example, convolution neural networks (CNN), originally proposed for image processing, are suitable structures for low-level subnets [3], [36].

CNNs realize filters as mathematical convolutions, i.e. the full input or a part of it is convoluted with the filter weights. These operations are inherently parallel processes such that

Figure 2. Schematic illustration of a deep network architecture. The ellipses represent either single-layer feedforward networks or layers.

Figure 3. Schematic illustration of a deep CNN.

parallel computer architectures can benefit from this structure. One prominent example is the GPU, which are designed to carry out respective mathematical calculation with a high degree of parallelism. Thus multilayer neural network structure consisting of CNN-layers as depicted in Fig. 3 became attractive during the last years making use of this advantage [7]. Another aspect of CNNs in deep architecture is that the learning of the filters is improved by the concept of shared weights, i.e. one reuses the same weights (filters) at each location of the input. This reduces the number of parameters, which makes networks easier to train. A convolution layer usually is followed by a pooling layer. The max-pooling frequently give superior results [27].

Yet, the last hierarchy module in a deep CNN (DCNN) is an SLFN or MLP using the cross entropy loss (5) [4]. Further, although the whole architecture is a supervised network, unsupervised training of single modules usually improves the learning stability and final network performance [35].

A serious problem in deep architectures is overfitting according to the huge number of trainable weights. Beside weight-sharing in deep CNN-networks several regularization schemes are known for deep architectures. Particularly dropout learning is one of the most appropriate methods, which simply drop units randomly during training [37].

C. Extreme Learning Machines

The extreme learning machine (ELM) as introduced in [38] is an SLFN, i.e. there is only one hidden layer $\mathbf{h}$. The matrix $\mathbf{W}^h$ connecting the input layer with the hidden layer according to (3) is initialized randomly and then kept constant all the time. Thus, the mapping (4) from the input $\mathbf{I}$ to the hidden layer $\mathbf{h} \in \mathbb{R}^{N_h}$ becomes a non-linear random projection. Thereby, different types of activation functions g_h can be considered

[39].¹ The mapping from the hidden layer to the output layer $\mathbf{O} \in \mathbb{R}^m$ is taken as a linear function such that we have simply

$$y_j = \langle \mathbf{w}_j^O, \mathbf{h} \rangle_E + \beta_j^O$$

where $\mathbf{w}_j^O$ is the j -th row vector of the matrix $\mathbf{W}^O \in \mathbb{R}^{m \times N_h}$.

Now we assume N training samples $(\mathbf{x}_k, \mathbf{t}_k)$ with data $\mathbf{x}_k \in \mathbb{R}^n$ and targets $\mathbf{t}_k \in \mathbb{R}^m$ for supervised training. We collect all hidden signals $\mathbf{H}_k = \mathbf{h}(\mathbf{x}_k)$ and target vectors in the matrices

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_1^T \\ \vdots \\ \mathbf{H}_N^T \end{pmatrix} \in \mathbb{R}^{N \times N_h} \text{ and } \mathbf{T} = \begin{pmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{pmatrix} \in \mathbb{R}^{N \times m}$$

such that the task to be solved by the ELM network becomes

$$(\mathbf{W}^O)^T = \min_{\mathbf{W} \in \mathbb{R}^{N_h \times m}} \|\mathbf{H} \cdot \mathbf{W} - \mathbf{T}\| \quad (9)$$

describing a linear optimization problem. Provided $N_h \ll N$ we get the solution

$$(\mathbf{W}^O)^T = \mathbf{H}^\dagger \cdot \mathbf{T}$$

where $\mathbf{H}^\dagger$ is the *Moore-Penrose-inverse* of $\mathbf{H}$ [38].

III. LEARNING VECTOR QUANTIZERS FOR PROTOTYPE BASED CLASSIFICATION

A. Standard Learning Vector Quantization

LVQ as introduced by KOHONEN are robust and intuitive nearest prototype classifiers [8], [9]. We assume data classes $1, \dots, C$ and data $\mathbf{x} \in X \subseteq \mathbb{R}^n$. The aim is to distribute a set $W = \{\mathbf{w}_1, \dots, \mathbf{w}_M\}$ of prototypes such that we can assign a class $c(\mathbf{x})$ to each $\mathbf{x} \in X$. In LVQ, each prototype $\mathbf{w}_k$ is equipped with a class labels $c(\mathbf{w}_k)$ such that at least one prototype is responsible for each class. Then the class assignment

$$c(\mathbf{x}) = c(\mathbf{w}_{s(\mathbf{x})})$$

for a data sample $\mathbf{x}$ is realized by means of a winner-take-all competition (WTAC)

$$s(\mathbf{x}) = \operatorname{argmin}_k (d(\mathbf{x}, \mathbf{w}_k)) \quad (10)$$

where d is a given dissimilarity measure [11]. We denote $\mathbf{w}_{s(\mathbf{x})}$ as the winner prototype of the competition. Frequently d is chosen as the squared Euclidean distance $d_E(\mathbf{x}, \mathbf{w}_k) = (\mathbf{x} - \mathbf{w}_k)^2$. A more general variant is the quadratic form

$$d_\Omega(\mathbf{x}, \mathbf{w}_k) = (\Omega \mathbf{x} - \mathbf{w}_k)^2 \quad (11)$$

with $\Omega \in \mathbb{R}^{n_p \times n}$ is the projection matrix and n_p is the projection dimension [41].²

¹ If the activation function g_h in (4) is linear and the dimensionality $1 \ll N_h < n$ it is the well-known linear random projection [40]. Otherwise, if $N_h \gg n$ it can be seen as a linear kernel mapping into a high-dimensional data space [22].

²Note that the quadratic form $d_\Omega(\mathbf{x}, \mathbf{w}_k)$ in (11) differs from the quadratic form

$$d_\Lambda(\mathbf{x}, \mathbf{w}_k) = (\mathbf{x} - \mathbf{w}_k)^T \Lambda (\mathbf{x} - \mathbf{w}_k)$$

frequently used in Generalized Matrix LVQ (GMLVQ) [42]. In this metric d_Λ , the matrix Λ is the so-called relevance matrix assumed to be positive definite. The entries of this matrix describe the correlations between the data features which contribute to a class discrimination. Thus one has $d_\Lambda(\mathbf{x}, \mathbf{w}_k) = (\hat{\Omega}(\mathbf{x} - \mathbf{w}_k))^2$ with $\Lambda = \hat{\Omega}^T \hat{\Omega}$ and $\hat{\Omega}$ projects both, data and prototypes. Thus the prototypes are living in the original data space whereas for (11) the prototypes are in the projection space $\mathbb{R}^{n_p}$.

Figure 4. Illustration of an LVQ as a two-layer neural network

We formally can take the LVQ network with the dissimilarity measure d_Ω as a multilayer network containing two hidden layers $\mathbf{h}^1$ and $\mathbf{h}^2$, see Fig. 4.

The nodes h_i^1 of the first hidden layer $\mathbf{h}^1 \in \mathbb{R}^{n_p}$ are associated with the row vectors ω_i of the projection matrix Ω according to

$$h_i^1(\mathbf{x}) = g_1(\langle \omega_i, \mathbf{x} \rangle_E + \beta_i^1)$$

where $\beta_i^1 = 0$ and $g_1(z) = \text{id}(z)$ is the identity function. Thus, the first layer realizes the projection of the data sample into the space $\mathbb{R}^{n_p}$ and is, therefore, denoted as the projection layer. The second layer $\mathbf{h}^2$ contains the prototypes $\mathbf{w}_i$. This layer is fully connected to the previous layer $\mathbf{h}^1$ by

$$h_j^2(\mathbf{x}) = g_2(d(\mathbf{h}^1(\mathbf{x}), \mathbf{w}_j)) \quad (12)$$

using an arbitrary (differentiable) dissimilarity measure d and g_2 is again the identity function. The connection to the output layer $\mathbf{O} \in \mathbb{R}^m$, with here having $m = C$ according to the number C of available classes, is realized as

$$O_l = \delta_{l,c(\mathbf{w}_{s(\mathbf{x})})} \quad (13)$$

where

$$\delta_{l,c} = \begin{cases} 1 & \text{if } l = c \\ 0 & \text{else} \end{cases}$$

is the Kronecker symbol and $s(\mathbf{x})$ is the winner index from WTAC (10). Thus, the last connections can be seen as a non-linear layer connectivity comparing the response of the prototypes from layer $\mathbf{h}^2$. Therefore, the second layer $\mathbf{h}^2$ is also denoted as competition layer.

If a kernel distance $d_\kappa(\mathbf{x}, \mathbf{w}_j)$ with kernel $\kappa(\mathbf{x}, \mathbf{w}_j)$ is used as a dissimilarity measure in LVQ the projection layer $\mathbf{h}^1$ would realize the kernel mapping $\Phi_\kappa(\mathbf{x})$. However, due to the kernel trick this mapping is not carried out explicitly.

Learning in LVQ takes place by distributing the prototypes in the projection space $\mathbb{R}^{n_p}$ of the projection layer to represent the class distributions. If $n_p = n$ is valid then the prototypes live in the original data space. KOHONEN introduced the following heuristic learning scheme to approximate a Bayesian

classifier [9]: Depending on the agreement of the class labels $c(\mathbf{x})$ of the data and the label $c(\mathbf{w}_{s(\mathbf{x})})$ of the winning prototype the prototype is moved towards $\mathbf{h}^1(\mathbf{x})$ or repelled from them. More precisely, we define the quantity

$$\Psi(\mathbf{x}, s(\mathbf{x})) = \begin{cases} 1 & \text{if } c(\mathbf{x}) = c(\mathbf{w}_{s(\mathbf{x})}) \\ -1 & \text{if } c(\mathbf{x}) \neq c(\mathbf{w}_{s(\mathbf{x})}) \end{cases}$$

such that the update becomes

$$\Delta \mathbf{w}_{s(\mathbf{x})} = \varepsilon \Psi(\mathbf{x}, s(\mathbf{x})) \cdot (\mathbf{\Omega} \mathbf{x} - \mathbf{w}_{s(\mathbf{x})})$$

realizing a vector shift. The value $0 < \varepsilon \ll 1$ denotes as learning rate. Note at this point that

$$\frac{\partial d_{\mathbf{\Omega}}(\mathbf{x}, \mathbf{w}_k)}{\partial \mathbf{w}_k} = -2(\mathbf{\Omega} \mathbf{x} - \mathbf{w}_k)$$

is valid if d in (12) is the quadratic form $d_{\mathbf{\Omega}}$ from (11).

Although this intuitive LVQ learning scheme is quite simple, the classification performance of this basic LVQ is often comparable to much more complicate methods. However, it is just a heuristic and does not realize a minimization scheme for the classification error. Therefore, variants of the basic scheme were developed minimizing at least an approximated classification error. Two of them are discussed in the following in greater detail.

B. The Generalized LVQ

The first approach presenting a cost function was proposed by SATO & YAMADA in [43]. This approach, denoted as generalized LVQ (GLVQ) takes as cost function

$$E_{GLVQ}(W) = \sum_{k=1}^N E(\mathbf{x}_k, W)$$

with

$$E(\mathbf{x}_k, W) = \varphi(\mu(\mathbf{x}_k, W))$$

describing the local errors. Here $\varphi(z)$ is a monotonically increasing function frequently chosen as the identity function $\text{id}(z) = z$ or the sigmoid function $\phi(z, \theta)$ from (1). The function

$$\mu(\mathbf{x}, W) = \frac{d(\mathbf{x}, \mathbf{w}^+) - d(\mathbf{x}, \mathbf{w}^-)}{d(\mathbf{x}, \mathbf{w}^+) + d(\mathbf{x}, \mathbf{w}^-)} \quad (14)$$

is the so-called classifier function. Here, $\mathbf{w}^+$ is the best matching prototype with the correct class label $c(\mathbf{x}) = c(\mathbf{w}_{s(\mathbf{x})})$ and $\mathbf{w}^-$ denotes the best matching prototype with an incorrect class label $c(\mathbf{x}) \neq c(\mathbf{w}_{s(\mathbf{x})})$. Thus $\mu(\mathbf{x}, W) \in [-1, 1]$ takes negative values if $\mathbf{x}$ is correctly classified. The dissimilarity function d has to be differentiable. The cost function E_{GLVQ} is an approximation of the overall classification error [44].

Learning in GLVQ usually takes place as SGDL for E_{GLVQ} according to

$$\Delta \mathbf{w}^{\pm} \propto -\xi(\mathbf{x}, \mathbf{w}^{\pm}) \cdot \frac{\partial \mu}{\partial d^{\pm}(\mathbf{x})} \frac{\partial d^{\pm}(\mathbf{x})}{\partial \mathbf{w}^{\pm}} \quad (15)$$

where the scaling factor

$$\xi(\mathbf{x}, \mathbf{w}^{\pm}) = \frac{\partial E(\mathbf{x}_k)}{\partial \varphi} \cdot \frac{\partial \varphi}{\partial \mu} \quad (16)$$

is obtained applying the chain rule for differentiation with the short hand notation $d^{\pm}(\mathbf{x}) = d(\mathbf{x}, \mathbf{w}^{\pm})$. We remark that the update (15) takes explicitly into account the derivative of the dissimilarity function used in GLVQ.

If the dissimilarity $d_{\mathbf{\Omega}}(\mathbf{x}, \mathbf{w}_k)$ from (11) is used in the classifier function μ from (14), the parameter $\mathbf{\Omega} \in \mathbb{R}^{n_p \times n}$ also can be adjusted by SGDL using the local errors

$$E(\mathbf{x}_k, W, \mathbf{\Omega}) = \varphi(\mu(\mathbf{x}_k, W, \mathbf{\Omega})) \quad (17)$$

with μ now depending also on $\mathbf{\Omega}$. The corresponding update becomes

$$\Delta \mathbf{\Omega} \propto -\xi(\mathbf{x}, \mathbf{w}^{\pm}) \cdot \frac{\partial \mu}{\partial \mathbf{\Omega}}$$

using the scaling factor $\xi(\mathbf{x}, \mathbf{w}^{\pm})$ from (16). Here we have

$$\frac{\partial \mu}{\partial \mathbf{\Omega}} = \frac{\partial \mu}{\partial d_{\mathbf{\Omega}}^+(\mathbf{x})} \cdot \frac{\partial d_{\mathbf{\Omega}}^+(\mathbf{x})}{\partial \mathbf{\Omega}} + \frac{\partial \mu}{\partial d_{\mathbf{\Omega}}^-(\mathbf{x})} \cdot \frac{\partial d_{\mathbf{\Omega}}^-(\mathbf{x})}{\partial \mathbf{\Omega}} \quad (18)$$

with

$$\frac{\partial \mu}{\partial d_{\mathbf{\Omega}}^+(\mathbf{x})} = \frac{-2d_{\mathbf{\Omega}}^-(\mathbf{x})}{(d_{\mathbf{\Omega}}^+(\mathbf{x}) + d_{\mathbf{\Omega}}^-(\mathbf{x}))^2} \quad (19)$$

and

$$\frac{\partial \mu}{\partial d_{\mathbf{\Omega}}^-(\mathbf{x})} = \frac{2d_{\mathbf{\Omega}}^+(\mathbf{x})}{(d_{\mathbf{\Omega}}^+(\mathbf{x}) + d_{\mathbf{\Omega}}^-(\mathbf{x}))^2} \quad (20)$$

as components of the derivative $\frac{\partial \mu}{\partial \mathbf{\Omega}}$. This GLVQ variant is known as GMLVQ [41], [42], [45]. If $n_p = n$ and $\mathbf{\Omega}$ is restricted to be a diagonal matrix, relevance learning GLVQ (GRLVQ) as introduced in [46] is recovered.

Recently, advanced stochastic gradient algorithms as AdaDelta, Adam and vSGDL are investigated for use in GMLVQ [47]. The two latter algorithms performed best in this study and, hence, are highly recommended.

C. Robust Soft LVQ and Probabilistic Robust Soft LVQ

Robust soft LVQ (RSLVQ) was introduced in [48] as a probabilistic variant for LVQ networks. RSLVQ assumes a data density $p(\mathbf{x})$, which is a mixture of the class conditional probabilities $p(\mathbf{x}|c)$, i.e.

$$p(\mathbf{x}) = \sum_{c=1}^C p(\mathbf{x}|c) P(c)$$

with $P(c)$ is the prior of class c . The class conditional probabilities $p(\mathbf{x}|c)$ are determined in RSLVQ based on the prototype set W , i.e. we have the estimators $p_W(\mathbf{x}|c)$ for $p(\mathbf{x}|c)$ as

$$p(\mathbf{x}, c|W) = \sum_{j=1}^M \delta_{c, c(\mathbf{w}_j)} p(\mathbf{x}|\mathbf{w}_j) P_W(\mathbf{w}_j)$$

where $P_W(\mathbf{w}_j)$ is the prior for the prototype $\mathbf{w}_j$ and $p(\mathbf{x}|\mathbf{w}_j)$ is the probability that prototype $\mathbf{w}_j$ has generated the data point $\mathbf{x}$. This latter probability is usually chosen to be of Gaussian shape

$$p(\mathbf{x}|\mathbf{w}_j) = \alpha_j \cdot \exp\left(\frac{-d(\mathbf{x}, \mathbf{w}_j)}{\sigma_j^2}\right)$$

with $d(\mathbf{x}, \mathbf{w}_j)$ being the data dissimilarity and σ_j is the width. Analogously, we have

$$p_W(\mathbf{x}|\bar{c}) = \sum_{j=1}^M (1 - \delta_{c, c(\mathbf{w}_j)}) p(\mathbf{x}|\mathbf{w}_j) P_W(\mathbf{w}_j)$$

as the probability that $\mathbf{x}$ is not generated by the class c according to the RSLVQ model.

A novel data point $\mathbf{x}$ is assigned to the class $c^*(\mathbf{x}|W)$ according to the rule

$$c^*(\mathbf{x}|W) = \operatorname{argmax}_{c=1, \dots, C} p(\mathbf{x}, c|W)$$

which can be approximated by the simpler WTAC rule (10) such that $c(\mathbf{w}_{s(\mathbf{x})})$ is the likeliest label for $c^*(\mathbf{x}|W)$. Hence, we obtain for the output layer $\mathbf{O} \in \mathbb{R}^C$ of the network

$$O_l = \delta_{l, c^*(\mathbf{x}|W)}$$

replacing the output determination (13) of standard LVQ and GLVQ. Alternatively, a *probabilistic decision* would be delivered by the RSLVQ network if we take

$$O_l = p(\mathbf{x}, l|W)$$

for $l = 1, \dots, C$.

Learning in RSLVQ takes place as an optimization of the negative log-likelihood

$$L_{RSLVQ}^- = - \sum_{k=1}^N \log \left(\frac{p_W(\mathbf{x}_k | c(\mathbf{x}_k))}{p(\mathbf{x}_k | W)} \right) \quad (21)$$

where

$$p(\mathbf{x}|W) = \sum_{j=1}^M p(\mathbf{x}|\mathbf{w}_j) P_W(\mathbf{w}_j)$$

is the class-independent model probability for generating $\mathbf{x}$ by the RSLVQ network. Here, we have the local negative log-likelihoods

$$L^-(\mathbf{x}_k) = - \log \left(\frac{p_W(\mathbf{x}_k | c(\mathbf{x}_k))}{p(\mathbf{x}_k | W)} \right)$$

such that the SGDL strategy applied to L_{RSLVQ}^- yields

$$\Delta \mathbf{w}_j \propto \begin{cases} (p_c(\mathbf{w}_j | \mathbf{x}_k) - p(\mathbf{w}_j | \mathbf{x}_k)) \cdot \frac{\alpha_j}{\sigma_j^2} \cdot \frac{\partial d(\mathbf{x}_k, \mathbf{w}_j)}{\partial \mathbf{w}_j} & \text{if } c(\mathbf{x}_k) = c(\mathbf{w}_j) \\ -p(\mathbf{w}_j | \mathbf{x}_k) \cdot \frac{\alpha_j}{\sigma_j^2} \cdot \frac{\partial d(\mathbf{x}_k, \mathbf{w}_j)}{\partial \mathbf{w}_j} & \text{if } c(\mathbf{x}_k) \neq c(\mathbf{w}_j) \end{cases}$$

as prototype updates [48]. Here, the probabilities

$$p_c(\mathbf{w}_j | \mathbf{x}_k) = \frac{P_W(\mathbf{w}_j) p(\mathbf{x}_k | \mathbf{w}_j)}{p(\mathbf{x}, c|W)}$$

and

$$p(\mathbf{w}_j | \mathbf{x}_k) = \frac{P_W(\mathbf{w}_j) p(\mathbf{x}_k | \mathbf{w}_j)}{p(\mathbf{x}|W)}$$

are used according to the Bayes' theorem.

If the dissimilarity d in RSLVQ is chosen as $d_\Omega(\mathbf{x}, \mathbf{w}_k)$ from (11) the matrix Ω can again adjusted according to SGDL as

$$\Delta \Omega \propto - \frac{\partial L^-(\mathbf{x}_k, \Omega)}{\partial \Omega}$$

with

$$-\frac{\partial L^-}{\partial \Omega} = \sum_{j=1}^M \delta_{c(\mathbf{x}_k), c(\mathbf{w}_j)} \cdot \frac{\alpha_j}{\sigma_j^2} \cdot \frac{\partial d_\Omega(\mathbf{x}_k, \mathbf{w}_j)}{\partial \Omega} \quad (22)$$

$$(p_c(\mathbf{w}_j | \mathbf{x}_k) - (2 - \delta_{c(\mathbf{x}_k), c(\mathbf{w}_j)}) p(\mathbf{w}_j | \mathbf{x}_k))$$

as proposed in [42]. We denote this matrix variant of RSLVQ as RSMLVQ.

An investigation of advanced SGDL schemes for RSLVQ is not known so far to our best knowledge.

IV. COMBINATION OF LVQ VARIANTS AND MLP/ELM/DEEP ARCHITECTURES

In this chapter we briefly describe possible combinations of LVQ networks with MLPs, ELMs and general deep architectures. For LVQ models we concentrate on GMLVQ/GLVQ and RSMLVQ/RSLVQ. In all these architectures outlined in the following the MLP/ELM/Deep architecture are viewed as filters as it was already suggested for signal processing in [49].

A. Fusion of LVQ networks with MLP networks

Now we suppose the MLP network realizes a map $\Phi(\mathcal{P}) : \mathbf{x} \mapsto \mathbf{y}_\Phi$, where $\mathcal{P}$ denotes the set of all adjustable parameters of the MLP.

1) *Fusion of GMLVQ with MLP networks:* Obviously, we can feed the output $\mathbf{y}_\Phi(\mathbf{x})$ of the MLP network into the input layer $\mathbf{I}_{GMLVQ}$ of a GMLVQ network. In this way we obtain a MLP-GMLVQ architecture where the local errors $E(\mathbf{x}_k, W, \Omega)$ from (17) or L_k from cross entropy loss E_{MLP-CE} (5) are now also depending on the parameters of MLP network, i.e. $E_{\Phi-GMLVQ}(\mathbf{x}_k) = E_{\Phi-GMLVQ}(\mathbf{x}_k, W, \Omega, \mathcal{P})$. Thus, SGDL for MLP-GMLVQ with respect to a parameter $p_i \in \mathcal{P}$ is obtained as

$$\Delta p_i \propto - \frac{\partial E_{\Phi-GMLVQ}(\mathbf{x}_k, W, \Omega, \mathcal{P})}{\partial p_i}$$

with

$$\frac{\partial E_{\Phi-GMLVQ}(\mathbf{x}_k, W, \Omega, \mathcal{P})}{\partial p_i} = \xi(\mathbf{y}_\Phi(\mathbf{x}), \mathbf{w}^\pm) \cdot \frac{\partial \mu}{\partial p_i}$$

where the derivative $\frac{\partial \mu}{\partial p_i}$ takes the form

$$\frac{\partial \mu}{\partial p_i} = \frac{-2d_\Omega^-(\mathbf{y}_\Phi)}{(d_\Omega^+(\mathbf{y}_\Phi) + d_\Omega^-(\mathbf{y}_\Phi))^2} \cdot \frac{\partial d_\Omega^+(\mathbf{y}_\Phi)}{\partial p_i} + \frac{2d_\Omega^+(\mathbf{y}_\Phi)}{(d_\Omega^+(\mathbf{y}_\Phi) + d_\Omega^-(\mathbf{y}_\Phi))^2} \cdot \frac{\partial d_\Omega^-(\mathbf{y}_\Phi)}{\partial p_i}$$

in complete analogy to (18)–(20). We can simply proceed by means of

$$\frac{\partial d_\Omega^\pm(\mathbf{y}_\Phi)}{\partial p_i} = \frac{\partial d_\Omega^\pm(\mathbf{y}_\Phi)}{\partial \mathbf{y}_\Phi} \cdot \frac{\partial \mathbf{y}_\Phi}{\partial p_i}$$

where the last derivative $\frac{\partial \mathbf{y}_\Phi}{\partial p_i}$ finally leads to the back-propagation learning rules of MLPs. In this way, all parameters

of the MLP can be adapted with respect to the new cost function $E_{\Phi-GMLVQ}(\mathbf{x}_k)$.

If the MLP is a deep architecture, this approach realizes a deep GMLVQ. A deep GLVQ according to this scheme was already suggested in [50]. However, the respective learning strategy was not explained explicitly.

We emphasize that in the (deep) MLP-GMLVQ the adaptation of the projection matrix Ω is still possible via gradient descent learning. Further we remark that in MLP-GMLVQ the MLP can be seen as a non-linear adaptive filter realizing an alternative to the information theoretic motivated MLP-filter learning proposed in [6], [49].

2) *Fusion of RSMLVQ with MLP networks*: As for GMLVQ we consider now the integration of MLP into RSMLVQ denoted as MLP-RSMLVQ. Thus, we have to analyze the *local* negative log-likelihoods according to (21) now written as

$$L_{\Phi}^{-}(\mathbf{y}_{\Phi}, \Omega, \mathcal{P}) = -\log \left(\frac{p_W(\mathbf{y}_{\Phi}|c(\mathbf{y}_{\Phi}))}{p(\mathbf{y}_{\Phi}|W)} \right)$$

regarding the parameters $\mathcal{P}$ of the MLP for given $\mathbf{y}_{\Phi}(\mathbf{x})$. Again we use the knowledge from the Ω -update (22) in RSMLVQ and write

$$-\frac{\partial L_{\Phi}^{-}}{\partial p_i} = \sum_{j=1}^M \delta_{c(\mathbf{x}), c(\mathbf{w}_j)} \cdot \frac{\alpha_j}{\sigma_j^2} \cdot \frac{\partial d_{\Omega}(\mathbf{y}_{\Phi}, \mathbf{w}_j)}{\partial p_i} \\ p_c(\mathbf{w}_j|\mathbf{y}_{\Phi}) - (2 - \delta_{c(\mathbf{x}), c(\mathbf{w}_j)}) p(\mathbf{w}_j|\mathbf{y}_{\Phi})$$

with

$$\frac{\partial d_{\Omega}(\mathbf{y}_{\Phi})}{\partial p_i} = \frac{\partial d_{\Omega}(\mathbf{y}_{\Phi})}{\partial \mathbf{y}_{\Phi}} \cdot \frac{\partial \mathbf{y}_{\Phi}}{\partial p_i}$$

as before for MLP-GMLVQ. Again, the adaptation of the projection matrix Ω according to (22) is still possible.

3) *Fusion of GMLVQ and RSLVQ with CNN*: The combination of GMLVQ with a set $\mathfrak{F} = \{\mathbf{f}_j\}$ of adaptive filters for image classification was introduced in [51]. Particularly, in this approach each prototype $\mathbf{w}_k$ is equipped with a local projection matrix Ω_k such that each image vector $\mathbf{x}$ is represented by a descriptor

$$\mathbf{r}_k(\mathbf{x}) = \mathbf{x} \Omega_k^T * \sum_j \mathbf{f}_j$$

serving as input for $\mathbf{w}_k$ in the GMLVQ network. In this network the prototypes $\mathbf{w}_k$, the local matrices Ω_k as well as the filters $\mathbf{f}_j$ can be adapted according to a gradient descent learning scheme.

Following this direction, the application of DCNN as an adaptive filter for GMLVQ becomes straightforward; because the last module after all convolution and sub-sampling layers is a SLFN or MLP, see Fig. 3. Hence, we can apply the idea from before, i. e. we feed the output of the MLP-module $\mathbf{y}_{\Phi DCNN}$ of the DCNN to the input of a GMLVQ-network. In this way, the computational advantages of DCNN can be combined with the interpretable GMLVQ model.

Obviously, both strategies can also be incorporated into the RSMLVQ model, if a probabilistic prototype based model is demanded.

B. Fusion of LVQ networks with ELM networks

For the fusion of ELM and LVQ we suppose that an ELM network realizes a generally non-linear map $\Phi_{ELM} : \mathbf{x} \rightarrow \mathbf{h} \in \mathbb{R}^{N_h}$ by means of (3) with a randomly chosen but fixed matrix $\mathbf{W}^h$. Instead of optimizing the output connection matrix $\mathbf{W}^o \in \mathbb{R}^{m \times N_h}$ solving the minimum problem (9), we suggest to take the vector $\mathbf{h}$ as input for LVQ networks with matrix learning. In this case the projection matrix Ω of GMLVQ realizes the map $\Omega : \mathbb{R}^{N_h} \rightarrow \mathbb{R}^{n_p}$. Obviously, no additional adjustment of the ELM is needed for LVQ optimization. Thus we just make use of the generally non-linear random projection of the data into the maybe very-high-dimensional space $\mathbb{R}^{N_h}$ by the ELM. Again we remark that if Φ_{ELM} is linear, we have the usual linear random projection, see footnote 1.

Recently, the combination of a DCNN and ELM to accelerate the learning process moved into the focus of research [52], [53], [54]. Obviously, we can adopt this idea also for integration of DCNN-ELM into LVQ networks like GMLVQ and RSMLVQ.

V. CONCLUSION

In this conceptual paper we outlined possibilities to combine powerful deep network architectures with interpretable learning vector quantization (LVQ) for classification learning. Particularly, we stressed the fact that LVQ approaches can be seen as specific variants of multilayer networks with a specific cost function. The corresponding network representations allow a description of the LVQ models as feedforward networks. We explicitly discussed possibilities to integrate (deep) multilayer architectures, convolutional networks as well as extreme learning machines into learning vector quantization networks. In such fused models, the deep architectures play the role of adaptive filters which are merged in a G(M)LVQ model extending the flexibility of standard G(M)LVQ significantly. Moreover, pre-trained modules of deep architectures can be used as it is already the case for standard feature extraction schemes. Yet, the interpretability of the prototypes in LVQ schemes is reduced because they now live in the feature space generated by the deep architecture. However, explicit feature extraction in advance would deliver the same problem.

Finally, the application of advanced stochastic gradient learning schemes is always highly recommended for those combined models to accelerate and stabilize the convergence process in during learning.

ACKNOWLEDGMENT

The authors would like to thank M. Kaden and D. Nebel from the University of Applied Sciences Mittweida for stimulating discussions and critical remarks.

REFERENCES

- [1] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA., 1974.
- [2] P.J. Werbos. Backpropagation and neurocontrol: A review and prospectus. In *Proc. Of International Joint Conference on Neural Networks (Washington)*, Vol. 1, pages 209–216, Washington, DC., 1989.

- [3] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–2002, 1980.
- [4] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.
- [5] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, May 2015.
- [6] J. C. Principe, J.W. Fischer III, and D. Xu. Information theoretic learning. In S. Haykin, editor, *Unsupervised Adaptive Filtering*. Wiley, New York, NY, 2000.
- [7] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- [8] Teuvo Kohonen. Learning Vector Quantization. *Neural Networks*, 1(Supplement 1):303, 1988.
- [9] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).
- [10] M. Biehl, B. Hammer, and T. Villmann. Prototype-based models in machine learning. *Wiley Interdisciplinary Reviews: Cognitive Science*, 7(2):92–111, 2016.
- [11] D. Nebel, M. Kaden, A. Bohnsack, and T. Villmann. Types of (dis-)similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, page in press, 2017.
- [12] M. Kaden, M. Lange, D. Nebel, M. Riedel, T. Geweniger, and T. Villmann. Aspects in classification learning - Review of recent developments in Learning Vector Quantization. *Foundations of Computing and Decision Sciences*, 39(2):79–105, 2014.
- [13] D. Nova and P.A. Estévez. A review of learning vector quantization classifiers. *Neural Computation and Applications*, 25(511–524), 2013.
- [14] T. Villmann, A. Bohnsack, and M. Kaden. Can learning vector quantization be an alternative to SVM and deep learning? *Journal of Artificial Intelligence and Soft Computing Research*, 7(1):65–81, 2017.
- [15] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.
- [16] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, New York, NY, 2006.
- [17] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, 2002.
- [18] T. Villmann, S. Haase, and M. Kaden. Kernelized vector quantization in gradient-descent learning. *Neurocomputing*, 147:83–95, 2015.
- [19] John A. Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*, volume 1 of *Santa Fe Institute Studies in the Sciences of Complexity: Lecture Notes*. Addison-Wesley, Redwood City, CA, 1991.
- [20] G.-B. Huang and H.A. Babri. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transaction on Neural Networks*, 9(1):224–229, 1998.
- [21] G.-B. Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transaction on Neural Networks*, 14(2):274–281, 2003.
- [22] B. Frénay and M. Verleysen. Parameter-insensitive kernel in extreme learning for non-linear support vector regression. *Neurocomputing*, 74(16):2526–2531, 2011.
- [23] I. Steinwart and A. Christmann. *Support Vector Machines*. Information Science and Statistics. Springer Verlag, Berlin-Heidelberg, 2008.
- [24] J.C. Principe. *Information Theoretic Learning*. Springer, Heidelberg, 2010.
- [25] T. Villmann and S. Haase. Divergence based vector quantization. *Neural Computation*, 23(5):1343–1392, 2011.
- [26] G. Deco and D. Obradovic. *An Information-Theoretic Approach to Neural Computing*. Springer, Heidelberg, New York, Berlin, 1997.
- [27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [28] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G.J. Gordon and D.B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, number 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.
- [29] G. Hinton, N. Srivastava, and K. Swersky. RMSprop: A mini-batch version of Rprop. In *Coursera course lecture 6 - Neural Networks for Machine Learning*. 2012.
- [30] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.
- [31] M.D. Zeiler. An adaptive learning rate method. Technical Report arXiv:1212.5701, arXiv preprint, 2012.
- [32] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, 2015, pages 1–13, 2015.
- [33] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning 2013*, volume 28, pages 343–351. Proceedings of Machine Learning Research, 2013.
- [34] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.
- [35] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, and P. Vincent. Why does unsupervised pre-training help deep learning. *Journal of Machine Learning Research*, 11:625–660, 2010.
- [36] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning non-linear combinations of kernels. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 396–404. Curran Associates, Inc., 2009.
- [37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [38] G.-B. Huang, Q.Y. Zhu, and C.K. Siew. Extreme learning machine: the theory and applications. *Neurocomputing*, 70(1):489–501, 2006.
- [39] G. Huang, G.-B. Huang, S. Song, and K. You. Trends in extreme learning machines: A review. *Neural Networks*, pages 32–48, 2015.
- [40] Samuel Kaski. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Proceedings of IJCNN’98, International Joint Conference on Neural Networks*, volume 1, pages 413–418. IEEE Service Center, Piscataway, NJ, 1998.
- [41] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26(1):159–173, 2012.
- [42] P. Schneider, B. Hammer, and M. Biehl. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21:3532–3561, 2009.
- [43] A. Sato and K. Yamada. Generalized learning vector quantization. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, pages 423–9. MIT Press, Cambridge, MA, USA, 1996.
- [44] M. Kaden, M. Riedel, W. Hermann, and T. Villmann. Border-sensitive learning in generalized learning vector quantization: an alternative to support vector machines. *Soft Computing*, 19(9):2423–2434, 2015.
- [45] P. Schneider, B. Hammer, and M. Biehl. Distance learning in discriminative vector quantization. *Neural Computation*, 21:2942–2969, 2009.
- [46] B. Hammer and T. Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9):1059–1068, 2002.
- [47] M. LeKander, M. Biehl, and H. deVries. Empirical evaluation of gradient methods for matrix learning vector quantization. In *Proceedings of the 12th Workshop on Self-Organizing Maps and Learning Vector Quantization (WSOM2017+)*, page in this volume. IEEE Press, 2017.
- [48] S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Computation*, 15:1589–1604, 2003.
- [49] D. Xu and J. Principe. Training MLPs layer-by-layer with the information potential. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN ’99*, pages 1045–1048, Los Alamitos, 1999. IEEE Press.
- [50] H. deVries, R. Memisevic, and A. Courville. Deep learning vector quantization. In M. Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN’2016)*, pages 503–508, Louvain-La-Neuve, Belgium, 2016. i6doc.com.
- [51] I. Giotis, K. Bunte, N. Petkov, and M. Biehl. Adaptive matrices and filters for color texture classification. *Journal of Mathematical Imaging and Vision*, 47(1):79–92, 2013.
- [52] J. Kim, J. Kim, G.-J. Jang, and M. Lee. Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection. *Neural Networks*, 87:109–121, 2017.
- [53] Y. Zeng, X. Xu, Y. Fang, and K. Zhao. Traffic sign recognition using extreme learning classifier with deep convolutional features. In *Proceedings 5th International Conference on Intelligence Science and Big Data Engineering. Image and Video Data Engineering, , ISIDE 2015, Suzhou (China)*, volume 1 of *LNCS 9242*, pages 272–280, Berlin-Heidelberg, 2015. Springer Verlag.
- [54] S. Pang and X. Yang. Deep convolutional extreme learning machine and its application in handwritten digit classification. *Computational Intelligence and Neuroscience*, 2016(6), 2016.