Generalized Learning Graph Quantization

Brijnesh J. Jain and Klaus Obermayer

Berlin Institute of Technology, Germany
jbj@cs.tu-berlin.de

Abstract. This contribution extends generalized LVQ, generalized relevance LVQ, and robust soft LVQ to the graph domain. The proposed approaches are based on the basic learning graph quantization (`lgq`) algorithm using the orbifold framework. Experiments on three data sets show that the proposed approaches outperform `lgq` and `lgq2.1`.

1 Introduction

Learning vector quantization (LVQ) as introduced by Kohonen [11] is a supervised learning algorithm for pattern classification. To classify patterns, LVQ applies the nearest neighbor rule using a condensed set of prototypes. Prototypes are learned by combining competitive learning with supervision. LVQ is easy to implement, runs efficiently, allows to control the complexity of the resulting classifier, naturally deals with multiclass problems, constructs an informative rather than a black-box model, and in many cases provides state of the art performance. Due to well-known shortcomings of LVQ and LVQ2.1 more sophisticated and powerful learning vector quantizers such as generalized LVQ [16], generalized relevance LVQ [4], and soft robust LVQ [17] have been devised.

LVQ and related methods have been originally devised for feature vectors equipped with the Euclidean metric. Extensions have been proposed, for example, for vectors with arbitrarily differentiable distance functions [5], for variable length and warped feature sequences [18], for strings [12], and for graphs [8].

For graphs, LVQ and LVQ2.1 have been extended to the corresponding learning graph quantization algorithms `lgq` and `lgq2.1` and comparable results to state-of-the-art methods have been reported [8]. These findings give rise to the question at issue, whether extensions of more powerful learning vector quantizers to the graph domain yield improved learning graph quantizers.

In this contribution, we extend generalized LVQ, generalized relevance LVQ, and robust soft LVQ to the domain of attributed graphs. The proposed approaches are based on the orbifold framework for graphs [6] and on `lgq` [8]. Experiments on three data sets of the IAM graph database [14] show that the proposed algorithms outperform `lgq` and `lgq2.1`.

2 Graph Orbifolds

This section introduces attributed graphs and represent them as point of some orbifold [1]. Most of this presentation including proofs of statements and claims is based on the structure space formalism proposed by [6].

Representation of Graphs. Let $\mathbb{E}$ be a d-dimensional Euclidean space. An *attributed graph* $X = (V, E, \alpha)$ consists of a set V of *vertices*, a set $E \subseteq V \times V$ of *edges*, and an *attribute function* $\alpha : V \times V \to \mathbb{E}$, such that $\alpha(i, j) \neq \mathbf{0}$ for each edge and $\alpha(i, j) = \mathbf{0}$ for each non-edge. Attributes $\alpha(i, i)$ of vertices i may take any value from $\mathbb{E}$.

For simplifying the mathematical treatment, we assume that all graphs are of order n, where n is chosen to be sufficiently large. Graphs of order less than n, say $m < n$, can be extended to order n by including isolated vertices with attribute zero. For practical issues, it is important to note that limiting the maximum order to some arbitrarily large number n and extending smaller graphs to graphs of order n are purely technical assumptions to simplify mathematics. For pattern recognition problems, these limitations should have no practical impact, because neither the bound n needs to be specified explicitly nor an extension of all graphs to an identical order needs to be performed. When applying the theory, all we actually require is that the graphs are finite.

A graph X is completely specified by its *matrix representation* $\boldsymbol{X} = (\boldsymbol{x}_{ij})$ with elements $\boldsymbol{x}_{ij} = \alpha(i, j)$ for all $1 \leq i, j \leq n$. Let $\mathcal{X} = \mathbb{E}^{n \times n}$ be the Euclidean space of all $(n \times n)$-matrices with elements from $\mathbb{E}$ and let $\mathit{\Pi}^n$ be the set of all $(n \times n)$-permutation matrices. For each $\boldsymbol{P} \in \mathit{\Pi}^n$ we define a mapping

$$\gamma_{\boldsymbol{P}} : \mathcal{X} \to \mathcal{X}, \quad \boldsymbol{X} \mapsto \boldsymbol{P}^{\top} \boldsymbol{X} \boldsymbol{P}.$$

Then $\mathcal{G} = \{\gamma_{\boldsymbol{P}} \,:\, \boldsymbol{P} \in \mathit{\Pi}^n\}$ is a finite group acting on $\mathcal{X}$. For $\boldsymbol{X} \in \mathcal{X}$, the *orbit* of $\boldsymbol{X}$ is the set defined by $[\boldsymbol{X}] = \{\gamma(\boldsymbol{X}) \,:\, \gamma \in \mathcal{G}\}$. The quotient set

$$\mathcal{X}_{\mathcal{G}} = \{[\boldsymbol{X}] \,:\, \boldsymbol{X} \in \mathcal{X}\}$$

consisting of all orbits is a *graph orbifold*. Its *orbifold chart* is the surjective continuous mapping

$$\pi : \mathcal{X} \to \mathcal{X}_{\mathcal{G}}, \quad \boldsymbol{X} \mapsto [\boldsymbol{X}]$$

that projects each matrix representation $\boldsymbol{X}$ to its orbit $[\boldsymbol{X}]$.

Suppose that $\boldsymbol{X}$ is a matrix representation of some attributed graph X. Then the orbit $[\boldsymbol{X}]$ consists of all possible matrices that represent X. By identifying the attributed graphs X with the orbits $[\boldsymbol{X}]$, we can regard graphs from $\mathcal{G}_{\mathcal{A}}$ as point of the graph orbifold $\mathcal{X}_{\mathcal{G}}$. The orbifold chart $\pi : \mathcal{X} \to \mathcal{X}_{\mathcal{G}}$ projects matrices $\boldsymbol{X}$ to the graphs X they represent.

For notational convenience, we identify $\mathcal{X}$ with $\mathbb{E}^N$, where $N = n^2$ and consider vector- rather than matrix representations of graphs. We obtain a *vector representation* $\boldsymbol{x}$ of graph X by concatenating the columns of a matrix $\boldsymbol{X}$ representing X. We write $\boldsymbol{x} \in X$ if $\boldsymbol{x} \in \mathcal{X}$ projects to $X \in \mathcal{X}_{\mathcal{G}}$ via the orbifold chart $\pi(\boldsymbol{x}) = X$.

Intrinsic Metric. The *intrinsic metric* of a graph orbifold $\mathcal{X}_{\mathcal{G}}$ is of the form

$$d(X, X') = \min \left\{ \|\boldsymbol{x} - \boldsymbol{x}'\|^2 \,:\, \boldsymbol{x} \in X, \boldsymbol{x}' \in X' \right\},$$

where $\|\cdot\|$ is the Euclidean distance on $\mathcal{X}$. We call a pair $(\boldsymbol{x}, \boldsymbol{x}') \in X \times X'$ with $\|\boldsymbol{x} - \boldsymbol{x}'\|^2 = d(X, X')$ an *optimal alignment* of X and X'. By $\mathcal{A}(X, Y)$ we denote the set of all optimal alignments of X and Y.

Suppose that $\boldsymbol{x} \in X$ is an arbitrary vector representation. Since $\mathcal{G}$ is a group, we have

$$d_{\boldsymbol{x}}(Y) = \min \left\{ \|\boldsymbol{x} - \boldsymbol{y}\|^2 \,:\, \boldsymbol{y} \in Y \right\} = d(X, Y).$$

By symmetry, we have $d_{\boldsymbol{y}}(X) = d(Y, X)$. Hence, the graph distance $d(X, Y)$ can be determined by fixing an arbitrary vector representation $\boldsymbol{x} \in X$ and then finding a vector representation $\boldsymbol{y}_*$ from Y that minimizes $\|\boldsymbol{x} - \boldsymbol{y}\|^2$ over all vector representations $\boldsymbol{Y} \in Y$ and vice versa.

Note that the intrinsic metric is not a artificial construction for analytical purposes but rather is based on a generalized concept of maximum common subgraph and therefore appears in different guises as a common choice of proximity measure for graphs [2, 3, 19].

Orbifold Functions. Suppose that $\mathcal{X}_{\mathcal{G}}$ is a graph orbifold with orbifold chart $\pi : \mathcal{X} \to \mathcal{X}_{\mathcal{G}}$. An *orbifold function* is a mapping of the form $f : \mathcal{X}_{\mathcal{G}} \to \mathbb{R}$. The *lift* of f is a function $\tilde{f} : \mathcal{X} \to \mathbb{R}$ satisfying $\tilde{f} = f \circ \pi$. The lift $\tilde{f}$ is invariant under group actions of $\mathcal{G}$, that is $\tilde{f}(\boldsymbol{x}) = \tilde{f}\,(\gamma(\boldsymbol{x}))$ for all $\gamma \in \mathcal{G}$.

An example of an orbifold function is the parametrized metric $d_{\boldsymbol{x}}$ with $\boldsymbol{x} \in X$. In what follows, we investigate local analytical properties of $d_{\boldsymbol{x}}$. The *lift* $\tilde{d}_{\boldsymbol{x}}$ of the function $d_{\boldsymbol{x}}$ is defined by

$$\tilde{d}_{\boldsymbol{x}} : \mathcal{X} \to \mathbb{R}, \quad \boldsymbol{y} \mapsto \min \left\{ \|\boldsymbol{x} - \boldsymbol{y}'\|^2 \,:\, \boldsymbol{y}' \in Y \right\}.$$

Certainly, the lift satisfies $\tilde{d}_{\boldsymbol{x}} = d_{\boldsymbol{x}} \circ \pi$ and is invariant under group actions of $\mathcal{G}$, that is $\tilde{d}_{\boldsymbol{x}}(\boldsymbol{y}) = \tilde{d}_{\boldsymbol{x}}\,(\gamma(\boldsymbol{y}))$ for all $\gamma \in \mathcal{G}$.

By lifting the distance function $d_{\boldsymbol{x}}$ to the Euclidean space $\mathcal{X}$, we are in the position to transfer analytical concepts such as differentiability and gradients to functions on graph orbifolds. We say, the function $d_{\boldsymbol{x}}$ is continuous (locally Lipschitz, differentiable, generalized differentiable) at point $Y \in \mathcal{X}_{\mathcal{G}}$ if its lift $\tilde{d}_{\boldsymbol{x}}$ is continuous (locally Lipschitz, differentiable, generalized differentiable in the sense of Norkin [13]) at some vector representation $\boldsymbol{y} \in Y$. This definition is independent of the choice of the vector representation that projects to Y.

As a minimizer of a set of continuously differentiable distance functions, the function $d_{\boldsymbol{x}}$ is generalized differentiable at any point Y. Though $d_{\boldsymbol{x}}$ is not differentiable, it is locally Lipschitz and therefore differentiable almost everywhere.

Gradients. Suppose that $d_{\boldsymbol{x}}$ is differentiable at Y. Then the lift $\tilde{d}_{\boldsymbol{x}}$ is differentiable at any vector representation that projects to Y. The gradient $\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})$ of $\tilde{d}_{\boldsymbol{x}}$ at $\boldsymbol{y}$ is of the form

$$\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y}) = -2(\boldsymbol{x} - \boldsymbol{y}_*)$$

where $(\boldsymbol{x}, \boldsymbol{y}_*) \in \mathcal{A}(X, Y)$ is an optimal alignment. Since $d_{\boldsymbol{x}}$ is differentiable at Y, the optimal alignment $(\boldsymbol{x}, \boldsymbol{y}_*)$ is unique. From

$$\nabla d_{\boldsymbol{x}}(\gamma(\boldsymbol{y})) = \gamma \left(\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y}) \right)$$

for all $\gamma \in \mathcal{G}$ follows that the gradients of $\tilde{d}_{\boldsymbol{x}}$ at $\boldsymbol{y}$ and $\gamma(\boldsymbol{y})$ are vector representations of the same graph. Hence, at differentiable points Y, the gradient of $d_{\boldsymbol{x}}(Y)$ at Y is defined by the projection

$$\nabla d_{\boldsymbol{x}}(Y) = \pi\left(\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})\right)$$

of the gradient $\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})$ at vector representation $\boldsymbol{y} \in Y$. Thus, the gradient of $d_{\boldsymbol{x}}$ at Y is a well-defined graph pointing to the direction of steepest ascent.

Generalized Gradients. Now suppose that $d_{\boldsymbol{x}}$ is generalized differentiable at Y. Then the lift $\tilde{d}_{\boldsymbol{x}}$ is generalized differentiable at any vector representation that projects to Y. The subdifferential $\partial \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})$ of $\tilde{d}_{\boldsymbol{x}}$ at $\boldsymbol{y}$ is a convex set containing

$$-2(\boldsymbol{x} - \boldsymbol{y}_*) \in \partial \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})$$

as generalized gradient, where $(\boldsymbol{x}, \boldsymbol{y}_*) \in \mathcal{A}(X, Y)$ is an optimal alignment. From

$$\partial d_{\boldsymbol{x}}(\gamma(\boldsymbol{y})) = \gamma\left(\partial \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})\right)$$

for all $\gamma \in \mathcal{G}$ follows that the subderivatives of $\tilde{d}_{\boldsymbol{x}}$ at $\boldsymbol{y}$ and $\gamma(\boldsymbol{y})$ project to the same subset of graphs. Hence, at generalized differentiable points Y, the subderivative of $d_{\boldsymbol{x}}(Y)$ at Y is defined by the projection

$$\partial d_{\boldsymbol{x}}(Y) = \pi\left(\partial \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})\right)$$

of the subderivative $\nabla \tilde{d}_{\boldsymbol{x}}(\boldsymbol{y})$ at an arbitrary vector representation $\boldsymbol{y} \in Y$. Thus, the subderivative of $d_{\boldsymbol{x}}$ at Y is well-defined and coincides with the gradient at differentiable points, that is $\partial d_{\boldsymbol{x}}(Y) = \{\nabla d_{\boldsymbol{x}}(Y)\}$.

3 Learning Graph Quantization

Learning graph quantization (`lgq`) aims at constructing a classifier $c : \mathcal{X}_{\mathcal{G}} \to \mathcal{C}$ that maps graphs from $\mathcal{X}_{\mathcal{G}}$ to class labels from a finite set $\mathcal{C}$. The classifiers are parameterized by a set of k prototypes $Y_1, \ldots, Y_k \in \mathcal{X}_{\mathcal{G}}$ with class labels $c_1, \ldots, c_k \in \mathcal{C}$. We predict the class label $c(X)$ of a new graph $X \in \mathcal{X}_{\mathcal{G}}$ by assigning it to the class label of the closest prototype according to the nearest neighbor rule. The goal of learning is to find a set of k prototypes that best predicts the class labels of graphs from $\mathcal{X}_{\mathcal{G}}$.

In the following, we first review `lgq` and `lgq2.1` as proposed in [8]. Then we extend GLVQ, GRLVQ, and RSLVQ to the domain of graph orbifolds.

3.1 LGQ

Suppose that $\mathcal{S} = \{(X_i, y_i)\}_{i=1}^{n} \subseteq \mathcal{X}_{\mathcal{G}} \times \mathcal{C}$ is a training set consisting of n input graphs $X_i \in \mathcal{X}_{\mathcal{G}}$ together with class labels $y_i \in \mathcal{C}$. The algorithm first

chooses k prototypes $\mathcal{Y} = \{(Y_j, c_j)\}_{j=1}^{k}$ such that each class is represented by at least one prototype. Next, during adaption, the algorithm randomly choses an example $(X, y) \in \mathcal{S}$ from the training set and modifies the closest prototype Y_X in accordance with the current example. The input graph X attracts its closest prototype Y_X if the class labels y of X and c_X of Y_X agree. Otherwise, if the class labels differ, the input X repels the closest prototype Y_X. To determine the closest prototype, `lgq` applies the nearest neighbor rule

$$Y_X = \arg\min_{Y \in \mathcal{Y}} \{d(X, Y)\}.$$

To update the closest prototype Y_X, the algorithm first selects an optimal alignment $(\boldsymbol{x}, \boldsymbol{y}_{\boldsymbol{x}}) \in \mathcal{A}(X, Y)$. Then it applies the standard LVQ update rule

$$\boldsymbol{y}_{\boldsymbol{x}} \leftarrow \begin{cases} \boldsymbol{y}_{\boldsymbol{x}} + \eta(\boldsymbol{x} - \boldsymbol{y}_{\boldsymbol{x}}) & : \quad y = c_x \\ \boldsymbol{y}_{\boldsymbol{x}} - \eta(\boldsymbol{x} - \boldsymbol{y}_{\boldsymbol{x}}) & : \quad y \neq c_x \end{cases},$$

where η is a monotonically decreasing learning rate following the guidelines of stochastic optimization. The updated vector representation projects to the updated graph prototype. This process continues until the procedure satisfies a termination criterion.

3.2 LGQ2.1

In contrast to `lgq`, the `lgq2.1` procedure updates the two closest prototypes Y_X^1 and Y_X^2 in accordance to the current training example $(X, y) \in \mathcal{S}$. The algorithm adapts the prototypes Y_X^1 and Y_X^2 if the following conditions hold:

1. Exactly one of both prototypes Y_X^1 and Y_X^2 has the same class label as X
2. The input graph X falls in a window around the decision border defined by

$$\frac{d\left(X, Y_X^2\right)}{d\left(X, Y_X^1\right)} > \frac{1-w}{1+w},$$

   where w is the relative width of the window.

For each prototype `lgq2.1` uses the same update rule as `lgq`.

3.3 Generalized LGQ

We use the following notations: Suppose that (X, y) is an arbitrary training example from $\mathcal{S}$. Let Y^+ be the closest prototype hat belongs to the same class as the current input X, and likewise let Y^- be the closest prototype that belongs to a different class from X. By c^+ and c^- we refer to the class labels of Y^+ and Y^-, respectively. As before, Y_X denotes the closest prototype of X and c_X denotes the class of Y_X.

Following [16], generalized learning graph quantization (`glgq`) aims at minimizing the cost function

$$E = \sum_{i=1}^{n} f(\mu(X_i)),$$

where $f : \mathbb{R} \to \mathbb{R}$ is a monotonically increasing function and $\mu(X)$ is a function, which is positive if the class labels of X and Y_X agree and negative otherwise. We assume that $L = f \circ \mu$ is generalized differentiable. Then we can minimize E using the incremental generalized gradient method

$$Y^+ \leftarrow Y^+ - \eta G^+ \tag{1}$$

$$Y^- \leftarrow Y^- + \eta G^-, \tag{2}$$

where $G^\pm \in \partial L$ is a generalized gradient of L at $Y^\pm$. As for feature vectors [16], we can show that `lgq` and `lgq2.1` are special cases of `glgq`.

Motivated by the robust and powerful performance of GLVQ for feature vectors, we choose

$$f(\mu) = \frac{1}{1 + \exp(-\mu)}$$

and

$$\mu(X) = \frac{d^+ - d^-}{d^+ + d^-},$$

where $d^+ = d(X, Y^+)$ and $d^- = d(X, Y^-)$. Then for any optimal alignment $(\boldsymbol{x}, \boldsymbol{y}^\pm) \in \mathcal{A}(X, Y^\pm)$ the vector representations

$$\boldsymbol{g}^+ = \frac{f'(\mu(X)) \cdot d^-}{\left(d^+ + d^-\right)^2} \left(\boldsymbol{x} - \boldsymbol{y}^+\right) \tag{3}$$

$$\boldsymbol{g}^- = -\frac{f'(\mu(X)) \cdot d^+}{\left(d^+ + d^-\right)^2} \left(\boldsymbol{x} - \boldsymbol{y}^-\right) \tag{4}$$

project to generalized gradients $G^\pm \in \partial L\left(Y^\pm\right)$ of L at $Y^\pm$.

3.4 Generalized Relevance LGQ

Generalized relevance learning graph quantization (`grlgq`) extends an idea proposed by [4] to graph orbifolds. Following [4], we replace the distance metric $d(X, Y)$ by a prototype-dependent scaled version

$$d_\Lambda(X, Y) = \min\left\{ \|\boldsymbol{x} - \boldsymbol{y}\|^2_{\boldsymbol{\lambda}} \,:\, \boldsymbol{x} \in X \right\},$$

where $\Lambda \in \mathcal{X}_\mathcal{G}$ is an attributed graph, $\boldsymbol{y} \in Y$ as well as $\boldsymbol{\lambda} \in \Lambda$ are arbitrary but fixed vector representation, and

$$\|\boldsymbol{x} - \boldsymbol{y}\|^2_{\boldsymbol{\lambda}} = \sum_{i=1}^{N} \lambda_i \left(x_i - y_i\right)^2$$

is the scaled version of the squared Euclidean distance. Then updating amounts in updating the prototypes according to eqns. (1) and (2) accompanied by updating the relevance graph according to the rule

$$\Lambda^+ \leftarrow \Lambda^+ - \eta_1 H^+$$

$$\Lambda^- \leftarrow \Lambda^- - \eta_1 H^-,$$

where $\Lambda^{\pm}$ is the relevance graph of $Y^{\pm}$ and $H^{\pm} \in \partial L(\Lambda^{\pm})$ is a generalized gradient of L at $\Lambda^{\pm}$. Let

$$\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, \ldots, a_n b_n)$$

denote the Schur product of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$. Suppose that $(\boldsymbol{x}, \boldsymbol{y}^{\pm}) \in \mathcal{A}(X, Y^{\pm})$ is an optimal alignment. Then vector representations of the form

$$\boldsymbol{g}^{+} = \frac{f'(\mu(X)) \cdot d^{-}}{\left(d^{+} + d^{-}\right)^2} \cdot \boldsymbol{\lambda} \circ \left(\boldsymbol{x} - \boldsymbol{y}^{+}\right) \tag{5}$$

$$\boldsymbol{g}^{-} = -\frac{f'(\mu(X)) \cdot d^{+}}{\left(d^{+} + d^{-}\right)^2} \cdot \boldsymbol{\lambda} \circ \left(\boldsymbol{x} - \boldsymbol{y}^{-}\right) \tag{6}$$

project to generalized gradients $G^{\pm} \in \partial L\left(Y^{\pm}\right)$ of L at $Y^{\pm}$. Furthermore, any vector representation

$$\boldsymbol{h}^{+} = f'(\mu) \frac{d^{-}}{\left(d^{+} + d^{-}\right)^2} \left(\boldsymbol{x} - \boldsymbol{y}^{+}\right)^2$$

$$\boldsymbol{h}^{-} = f'(\mu) \frac{d^{+}}{\left(d^{+} + d^{-}\right)^2} \left(\boldsymbol{x} - \boldsymbol{y}^{-}\right)^2,$$

projects to a generalized gradient $H^{\pm} \in \partial L(\Lambda^{\pm})$. Observe that the update rule (5) and (6) of `grlgq` differs from the update rule (3) and (4) of `glgq` by including the relevance factors.

3.5 Robust Soft LGQ

Robust soft learning graph quantization (`rslgq`) is motivated by RSLVQ [17], which in difference to the other `lgq` aims at describing the distribution of the data by a Gaussian mixture model. The approach is to maximize the ratio L_r of the probability, that an example $(X, y) \in \mathcal{S}$ is generated by components of the model corresponding to those prototypes with a class label equal to y, and the probability, that the whole model generates X.

To extend RSLVQ to graph orbifolds, we assume that $(\boldsymbol{x}_j, \boldsymbol{y}_j) \in \mathcal{A}(X, Y_j)$ are optimal alignments of a given input graph X and the prototypes Y_j for $j \in \{1, ..., k\}$. The update rule for Y_j is then of the form

$$Y_j \leftarrow Y_j + \frac{\eta}{\sigma^2} G_j,$$

where

$$\boldsymbol{g}_j = \begin{cases} (P_{\boldsymbol{y}}(\boldsymbol{y}_j|\boldsymbol{x}_j) - P(\boldsymbol{y}_j|\boldsymbol{x}_j))(\boldsymbol{x}_j - \boldsymbol{y}_j), & : \quad \boldsymbol{y} = c_j \\ -P(\boldsymbol{y}_j|\boldsymbol{x}_j))(\boldsymbol{x}_j - \boldsymbol{y}_j), & : \quad \boldsymbol{y} \neq c_j \end{cases}$$

projects to a generalized gradient $G_j \in \partial \log(L_r(Y_j))$ and

$$P_{\boldsymbol{y}}(\boldsymbol{y}_j|\boldsymbol{x}_j) = \frac{\exp\left(-\dfrac{(\boldsymbol{x}_j - \boldsymbol{y}_j)^2}{2\sigma^2}\right)}{\displaystyle\sum_{i:c_i=\boldsymbol{y}} \exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{y}_i)^2}{2\sigma^2}\right)},$$

and

$$P(\boldsymbol{y}_j|\boldsymbol{x}_j) = \frac{\exp\left(-\frac{(\boldsymbol{x}_j - \boldsymbol{y}_j)^2}{2\sigma^2}\right)}{\sum_{i=1}^{k} \exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{y}_i)^2}{2\sigma^2}\right)}.$$

It is important to note that the probabilistic interpretation of RSLVQ is no longer valid for its counterpart in graph orbifolds. A first step to remove this shortcoming is presented in [10].

4 Experiments

We conducted first experiments to compare the performance of the different `lgq` algorithms.

4.1 Data.

We selected the following data sets from the IAM graph database repository: letter, grec, and fingerprint. Each data set is divided into a training, validation, and a test set. Table 1 provides a summary of the main characteristics of the data sets. For further details we refer to [14].

4.2 Experimental Setup

Setting of `lgq` algorithms. Given a data set, each `lgq` algorithm was first initialized with a single prototype for each class. To initialize the prototypes we computed a Frechet sample mean of all class members from the training set by using the incremental sample mean algorithm proposed in [7]. Next, we performed a parameter selection for the `lgq` algorithms. For each parameter configuration, we learned the prototypes using the training set and tested the learned model on both, the training and validation set. We selected the parameters that gave the best classification accuracy on the training and validation set. Finally, we assessed the generalization performance by applying the learned model on the test set. For all `lgq` algorithms we tuned the learning rate η. For `lgq2.1`, `grlgq`, and `rslgq`, we additionally calibrated the window width w, the learning rate η_λ of the relevance factors, and the width σ of the Gaussian. respectively.

Graph Distance Calculations and Optimal Alignment. For graph distance calculations and finding optimal alignments, we applied the extended Bron Kerbosch algorithm [9] with clique selection and $10\,|V_Z|$ as the maximum number of recursive calls, where V_Z denotes the vertex set of the association graph under consideration.

Protocol. All `lgq` algorithms have been applied to the training set of each data set 3 times. To assess the generalization performance on the test sets, we have chosen the model that best predicts the class labels on the training and validation

data set	#(classes)	avg(nodes)	max(nodes)	avg(edges)	max(edges)
letter (750, 750, 750)	15	4.7	8	3.1	6
grec (286, 286, 528)	22	11.5	24	11.9	29
fingerprint (500, 300, 2000)	4	8.3	26	14.1	48

Table 1. Summary of main characteristics of the data sets. The tiny numbers in parentheses show the size of the training, validation, and test set, respectively.

set. We compared the `lgq` algorithms with the similarity kernel in conjunction with the SVM (`sk+svm`) and the family of Lipschitz embeddings in conjunction with SVM (`ls+svm`) proposed by [15]. As a reference, we used the `knn` method based on the intrinsic metric, where the parameter k has been learned using the training and validation set.

4.3 Results

Table 2 summarizes the results. Since `sk+svm` and `le+svm` refer to a family of related methods rather than a single method, Table 2 presents the best result on the test set over all methods of the `sk+svm` and `le+svm` family for each data set. In doing so, the comparison is optimistically biased towards `sk+svm` and `le+svm`.

The first observation to be made is that the novel extensions, `glgq`, `grlgq`, and `rslgq` outperform `lgq` and `lgq2.1`. These finding are in line with results of LVQ algorithms for feature vectors. A fair comparison of `glgq`, `grlgq`, and `rslgq`, however, is difficult since the performance of any of the `lgq` variants critically depends on the proper choice of the parameters. An extensive parameter selection is only manageable for `lgq` (η), `glgq` (η) and to a certain extent also for `lgq2.1` (η, w). For `grlgq` (η, η_λ) and `rslgq` (η, σ), however, exploring the parameter space is comparatively too time consuming for two reasons: (i) for a given learning rate, `grlgq` and `rslgq` require more iterations during learning until convergence than the other three algorithms, and (ii) both, `grlgq` and `rslgq`, critically depend on two rather than one parameter as this is the case for `lgq` and `glgq`.

The second observation to be made is that all novel extensions of `lgq` are comparable to state-of-the-art solutions. All `lgq` variants, however, are computationally faster than `knn`, `sk+svm` and `le+svm`. The largest portion of the computational effort to classify an unseen graph X is attributable to calculating (or approximating) graph distances between X and a set of prototypes specified by the respective classifier. While the prototype set for `knn` consists of the whole training set, `sk+svm` and `le+svm` use about $40\% - 60\%$ of the training set as prototypes. In contrast, the number of prototypes of the `lgq` algorithms in this setting corresponds to the number of classes (15 letters, 22 grec, 4 fingerprint).

5 Conclusion

Extensions of GLVQ, GRLVQ, and RSLVQ to the domain of graphs outperform `lgq` and `lgq2.1`, provide state-of-the-art solution even if using a single prototype

	knn	sk+svm	le+svm	lgq	lgq2.1	glgq	grlgq	rslgq
letter	82.0	79.1	92.5	81.5	85.7	88.4	86.5	87.3
grec	96.8	94.9	96.8	86.2	92.6	97.5	97.0	97.4
fingerprint	80.0	41.0	82.8	79.9	81.5	84.8	84.0	84.1

Table 2. Classification accuracy (in %) of the `lgq` algorithms.

for each class, and are superior than `knn`, `sk+svm`, and `le+svm` with respect to run time during classification. In a practical setting, we recommend to use `glgq` because of its simplicity and excellent performance. Future work aims at applying the `lgq` algorithms to other data sets and exploring their performance with more than one prototype per class.

References

1. J.E. Borzellino, "Riemannian geometry of orbifolds", *PhD thesis*, University of California, Los Angelos, 1992.
2. T. Cour, P. Srinivasan, J. Shi, "Balanced graph matching", *NIPS*, 2006.
3. S. Gold, A. Rangarajan, "Graduated Assignment Algorithm for Graph Matching", *IEEE Transactions on PAMI*, 18:377–388, 1996.
4. B. Hammer, T. Villmann, "Generalized relevance learning vector quantization", *Neural Network* 15:1059–1068, 2002.
5. B. Hammer, M. Strickert, T. Villmann, "Supervised neural gas with general similarity measure", *Neural Processing Letters* 21(1):21–44, 2005.
6. B. Jain, K. Obermayer, "Structure Spaces", *Journal of Machine Learning Research*, 10:2667–2714, 2009.
7. B. Jain, K. Obermayer. "Algorithms for the Sample Mean of Graphs", *CAIP*, 2009.
8. B. Jain, S.D. Srinivasan, A. Tissen, K. Obermayer. "Learning Graph Quantization", *S+SSPR*, 2010.
9. B. Jain, K. Obermayer. "Extending Bron Kerbosch for Solving the Maximum Weight Clique Problem", *arXiv:1101.1266v1*, 2011.
10. B. Jain, K. Obermayer. "Maximum Likelihood for Gaussians on Graphs", *GbR*, 2011.
11. T. Kohonen, *Self-organizing maps*, Springer, 1997.
12. T. Kohonen, P. Somervuo, "Self-organizing maps of symbol strings", *Neurocomputing*, 21(1-3):19–30, 1998.
13. V.I. Norkin. Stochastic generalized-differentiable functions in the problem of nonconvex nonsmooth stochastic optimization, *Cybernetics*, 22(6), 804-809, 1986.
14. K. Riesen, H. Bunke, "IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning", *SSPR*, 2008.
15. K. Riesen, H. Bunke, "Graph Classification by Means of Lipschitz Embedding", *IEEE Transactions on Systems, Man, and Cybernetics*, 39(6):1472–1483, 2009.
16. A. Sato, K. Yamada, "Generalized learning vector quantization", *NIPS*, 1996.
17. S. Seo, K. Obermayer, "Soft learning vector quantization", *Neural computation*, 15(7):1589–1604, 2003.
18. P. Sumervuo, T. Kohonen, "Self-organizing maps and learning vector quantization for feature sequences", *Neural Processing Letters*, 10(2):151–159,1999.
19. S. Umeyama, "An eigendecomposition approach to weighted graph matching problems", *IEEE Transactions on PAMI*, 10(5):695–703, 1988.