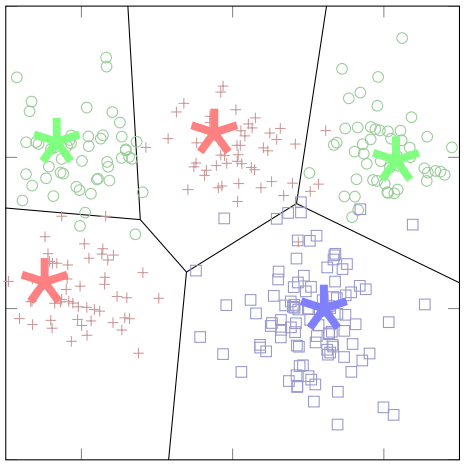
ON THE OPTIMIZATION OF GENERALIZED MATRIX LEARNING VECTOR QUANTIZATION

HARM DE VRIES



Msc.
Johann Bernoulli Institute for Mathematics and Computer Science
Rijksuniversiteit Groningen

February 2014

ACKNOWLEDGMENTS

This thesis brings an end to a wonderful time in Groningen. I am grateful to everybody who have supported me in completing my studies.

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Michael Biehl for his invaluable guidance throughout my studies. His broad technical knowledge and open-mindedness made meetings very helpful and stimulating. Michael was always available for advise, but also gave me the freedom to develop and pursue my own ideas.

Gratitude is also due to Nicolai Petkov, head of the Intelligent Systems group, who provided financial support for the Mittweida Workshops on Computational Intelligence (MIWOCI). I would like to thank Prof. Thomas Villmann for inviting me to this workshop and for giving me the opportunity to present my ideas to an excellent group of researchers. I am especially grateful for inspiring interactions with Prof. Barbara Hammer and Dr. Marc Strickert. Marc showed me around in Mittweida and influenced my thinking about the thesis topic by numerous discussions and e-mail conversations.

I would like to thank Prof. Paris Avgeriou for being my second supervisor. Also, a sincere thanks is due to my fellow students Zhe Sun, Herbert Kruitbosch, Martien Scheepens and Mark Scheeve for being intelligent people to work with.

Specials thanks goes to my room mates (and friends) Evert, Marie Suzanne, Fleur and Pieter for providing an absolutely amazing atmosphere at home. In particular, I would like to thank my best friend Pieter, with whom I share the same sense of humor, for joining me in so many activities over the last four years (ranging from cycling races to festivals). Also, I am thankful to my dear friends Emile, Jaap, Ruben, Peter, Ella, Samira for discussions about life.

Finally, I want to give specials thanks to my family for their unconditioned support. My mother always supported me in chasing my dreams, although it is hard for her to understand what I am exactly doing. I thank my father for being a good listener in difficult times, and for evoking my interest in mathematics. I also thank my younger brother Jaap for being there when I needed it most, and my older brother Evert for his constant belief in me.

CONTENTS

1 INTRODUCTION

Machine learning is a scientific discipline that concerns the study of systems that *learn* from data. Learning in this context is hard to define precisely, but in general we might say that a machine learns from experience whenever it adapts its system in such a manner that its expected future performance improves. Part of the reason for this coarse definition is that machine learning borrows techniques from many different fields. While most machine learning tasks are phrased as pure optimization problems [36], such techniques can be expressed in terms of statistical inference [39] and have biological interpretations [1].

A first question that needs to be answered: why should machines have to learn? Why not design machines that execute a task as desired in the first place? The reason is that for some tasks we do not have enough explicit knowledge about the problem to write a computer program to solve the task at hand. For example, consider the recognition of handwritten characters, that is, converting an image of handwritten text to ASCII text. Handwritten text depends on the person's handwriting style which may even change with the mood of the writer. Humans can, however, easily recognize such handwritten characters, but are not able to explain how they do it. This means that we dot not know the exact details of the process that generated the handwritten characters, and therefore we can not derive a set of simple rules to perform the recognition task.

In machine learning we typically take the following approach to solve the recognition task. We collect a large number of examples that specify the correct ASCII character for a given input image of a handwritten character. For each input image we then extract numerical *features* that are helpful to discriminate between the characters e.g. the ratio of the height of the character to its width. Finally, a learning algorithm generates a function that maps features of handwritten characters to the ASCII symbols. Learning algorithms differ in the employed model-class, that is the family of functions it can generate, and how it measures the discrepancy between the actual output of the model and the target output. *Learning* in these algorithms is defined as an optimization problem over the parameters of the model-class such that we minimize the discrepancy between the actual output of model and the target output on the collected data.

Last example of character recognition is an instance of supervised learning, in which we work with labeled data i.e. pairs of desired output for a given input. We speak of a classification problem when

the desired output is a class label, while in regression the desired output can take any real value. The other two broad categories of machine learning are unsupervised and reinforcement learning. Unsupervised learning algorithms do not need any form of external supervision and aim at discovering good internal representations of the data. Reinforcement learning concerns the finding of actions to maximize some notion of reward. For an extensive overview of all topics addressed in machine learning we refer to Alpaydin [1], Duda et al. [15].

In this thesis we focus on a recently proposed classification algorithm called Generalized Matrix Learning Vector Quantization (GMLVQ) [31]. The model-class of GMLVQ is parameterized by a set of prototypes, which are typical representatives of the classes, and a quadratic pseudo-metric. Classification is performed by a so called nearest prototype classification scheme i.e. a new input pattern is assigned to the class of the nearest prototype with respect to the learned quadratic pseudo-metric. Learning aims at error minimization and margin maximization at the same time and is formulated as an optimization problem over two qualitatively different groups: prototype and metric variables. The interplay between prototypes and the quadratic pseudo-metric turns out to be a challenging optimizing problem with interesting properties which will be highlighted throughout this thesis.

1.1 SCOPE

The purpose of this thesis is two-fold.

The first part concerns the understanding of the dynamics of GMLVQ. We aim to discover the characteristics of the outcome of the training procedure by thoroughly analysing the cost function and its constraints. We hereby address the following questions:

- What is the effect of the different terms in the cost function?
- Does the algorithm have any tendencies to learn a particular set of prototypes or quadratic metric?
- Is the constraint in the optimization problem really necessary?

In this light, we also investigate the first-order optimality conditions for the corresponding optimization problem. We especially focus whether the obtained solution is unique. In case it is not, we aim to present a unifying treatment of the ambiguities, and present additional constraints in order to obtain an interpretable solution.

The second part compares the performance of several optimization methods that work out of the box i.e. solvers that do not require a user to carefully tune any hyper parameters. This decision is motivated by the fact that the popularity of a machine learning algorithms is greatly

influenced by its ease of use for a standard user. For instance, the support vector machine (SVM) has gained popularity over the years due to its convex optimization problem, for which efficient and easy-to-use packages exist [13]. In this thesis we compare the performance of a recently proposed method that automatically adapts the learning rates for a stochastic gradient descent method, and a (limited) BFGS algorithm as a popular and robust example of a second-order batch method. In this study we aim to address the following questions:

- There is an ongoing debate whether second-order batch or stochastic gradient descent methods are most appropriate to optimize machine learning problems. What is the best choice in the context of GMLVQ?
- How does the adaptive learning rates for SGD perform in comparison with manually tuned learning rates?

1.2 ORGANIZATION

The remaining chapters of this thesis are organised as follows.

Chapter 2 provides background on Generalized Matrix Learning Vector Quantization (GMLVQ). We give a brief overview of the development of Learning Vector Quantization (LVQ) that eventually led to the proposal of GMLVQ. We formally introduce prototypes and the quadratic distance measure, and phrase the training procedure as an optimization problem.

Chapter 3 investigates the stationarity conditions of the optimization problem, that is, we work out the mathematical form of optimal prototypes and the optimal transformation matrix. The main finding is that prototypes can be formulated as a linear combination of the data points, and that the transformation matrix has a tendency to become low-rank. The results have implications for the uniqueness of the classifier i.e. several configurations of the prototypes and transformation matrix exist that result in a similar classifier. We thoroughly explain and illustrate these ambiguities, and propose additional constraints to obtain a unique and interpretable solution. Finally, we argue that all constraints can be removed from the optimization problem, since it is possible to project after training onto the feasible set.

Chapter 4 compares the performance of stochastic gradient descent, adaptive learning rates for stochastic gradient descent and (limited) BFGS on the learning problem of GMLVQ. We demonstrate that (limited) BFGS outperforms the stochastic optimization methods in terms of generalization performance and computational cost. Furthermore, BFGS should be preferred in low-dimensional datasets, while l-BFGS is the best choice for high-dimensional datasets. The adaptive learning rates for stochastic gradient descent turns out to be fairly conser-

vative, and only the individual learning rate variant, called vSGD-l, is close in performance to plain SGD.

2

GENERALIZED MATRIX LEARNING VECTOR QUANTIZATION

2.1 INTRODUCTION

Learning Vector Quantization (LVQ) is a popular family of supervised classification algorithms introduced by Kohonen in 1986 [21]. The approach has attracted much attention over the last twenty-five years because it is easy to implement, has a natural extension to multi-class problems and the resulting classifier is interpretable. A LVQ classifier is parametrized by a set of prototypes which serve as typical representative of the classes. The prototypes are determined in a training process from labelled example data and can be interpreted in a straightforward way since they reflect the characteristics of a class-specific cluster of data points. This is in contrast with many other 'black-box'classification algorithms, e.g. Support Vector Machines (SVM) and feed-forward Neural Networks, that lack a direct interpretation of their parameters. Fig. 1 shows an example of a LVQ classifier with five interpretable prototypes that all represent a class-specific cluster of data points.

Classification is performed by an intuitive nearest prototype classification scheme i.e. an unseen input object is assigned to the class of the nearest prototype. This classification scheme is closely related to the widely used k-Nearest Neighbour (k-NN) classifier [15], which assigns a new input object to the majority class among its k closest data points. One serious drawback of k-NN is that it requires to store and iterate over all training examples in order to classify an object. For large datasets this might become intractable in terms of memory usage and computation time. LVQ algorithms overcome these problems by using the training data to learn a set of *prototypical examples* that represent class-specific clusters of data points. This reduces the computational cost and memory usage to scale linearly in the number of prototypes instead of the number of data points. A LVQ classifier is faster than k-NN because the set of prototypes, which is specified by the user, is typically small and should depend on the the unknown number of class-specific clusters rather than the size of the dataset.

The originally proposed LVQ, called LVQ1 [21], learns the set of prototypes from the training data by means of heuristic Hebbian update rules. A randomly chosen training example is presented to the system and the closest prototype is pushed towards the considered example if it has the correct class, and is pushed away if the classes do not match. The magnitude of these prototype adjustments are con-

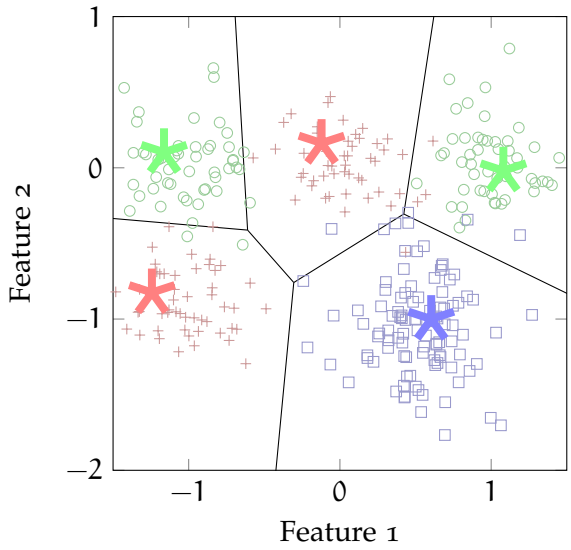
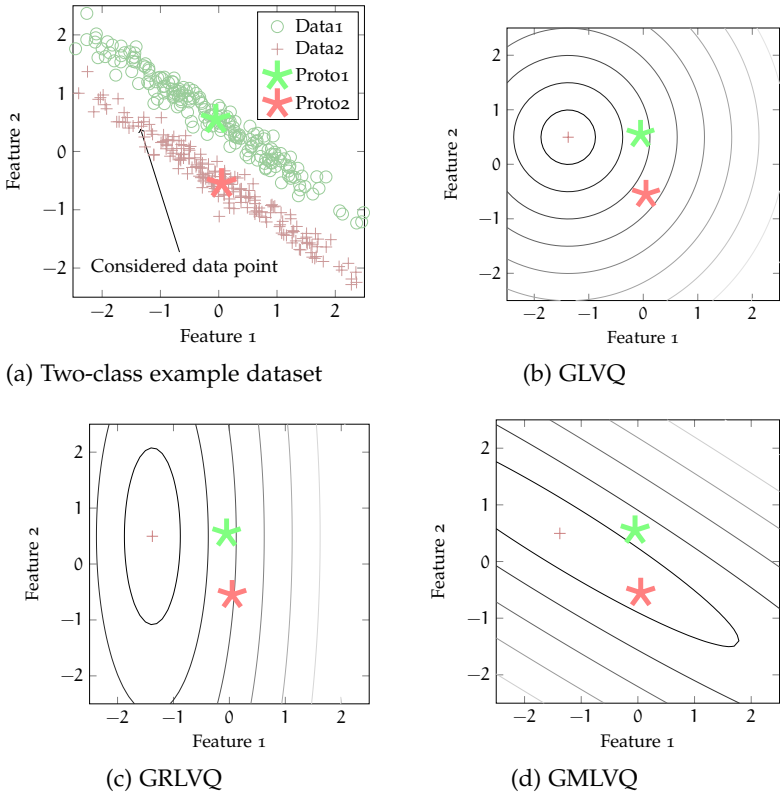


Figure 1: An example of a two-dimensional dataset with three classes. Data points belonging to class 1, class 2 and class 3 are marked by green '○', red '+' or blue '□' respectively. In this setting we have five prototypes, marked by '★' in the color of their class, which represent a class-specific cluster. The prototypes in combination with a distance measure(here the Euclidean distance) determine the decision boundaries which are rendered by black lines.

trolled by a learning rate, which should be carefully chosen to ensure convergence of the prototypes. Kohonen soon proposed an alternative update rule, called LVQ2.1 [22], that aims at better approximation of the Bayesian decision boundary in order to obtain faster convergence. In each training step two prototypes are updated: the closest correct prototype is moved towards the considered training example whilst the closest wrong prototype is pushed even further away. This update is, however, only performed when the considered example is in the vicinity of the decision boundary. Such a restriction is necessary to prevent divergent behavior i.e. prototypes that are pushed far away from the data.

LVQ variants that are motivated by heuristics limits the theoretical analysis of their dynamics and generalization behavior. Several LVQ variants were proposed that are derived from an explicit cost function [33, 28, 34]. Sato and Yamada introduced a prominent example that is formulated as a stochastic gradient descent procedure over a cost function which is now known as Generalized Learning Vector Quantization (GLVQ) Sato and Yamada [28]. By a stability analysis they show that, in contrast to LVQ2.1, their proposed cost function does not display divergent behavior. [14] derived worst case generalization bounds for prototype based classifiers using the framework of statistical learning theory. These bounds are in general rather loose since they hold for every possible data distribution, but they reveal



(a) Two-class example dataset (b) GLVQ

(c) GRLVQ (d) GMLVQ

Figure 2: In a) we show an example of a two-class dataset in which feature 1 and 2 are correlated. We pick a data point from the north-west corner of class 2. Considering this example as center we show in b-d) the equidistant-lines based on an instantiation of the employed distance metric in GLVQ, GRLVQ and GMLVQ, respectively.

the important factors that influence the generalization performance. Interestingly, the bound scales inversely with the so-called hypothesis margin i.e. the distance the prototypes can travel without changing the classification of the data points. In [14] it is pointed out that the learning objective of several LVQ variants is exactly maximization of this margin, and thus good generalization behavior can be expected. Moreover, the derived generalization bound does not depend on the input dimension which does not only imply good generalization performance for high dimensional datasets, but also opens up the possibility to meaningfully apply kernel techniques [20, 40]. Lastly, it is shown that the bound deteriorates when the number of prototypes increases which suggest that GLVQ is not only a faster classifier than its counterpart 1-NN, but also more accurate.

The performance of GLVQ crucially depend on the appropriateness of the Euclidean metric for the classification task at hand. For instance, this already fails for data sets in which features are unequally scaled or correlated. Although it is possible to choose any differentiable similarity measure for GLVQ to enhance performance, this will require

prior knowledge of the data set which is often not available. Hammer and Villmann were the first to propose an adaptive metric in the context of GLVQ. Their approach, called Generalized Relevance Learning Vector Quantization (GLRVQ), employs an adaptive weighted Euclidean metric which allows for scaling of the dimensions. This thesis focus on a further extension, Generalized Matrix Learning Vector Quantization (GMLVQ) [31], that employs a quadratic metric with a full adaptive matrix that can account for pairwise correlation between features. In Fig. 2 we show an intuitive example why GMLVQ might improve the performance of the classifier over GLVQ and GRLVQ. Consider the two-class labelled dataset shown in Fig. 2a in which the features are obviously correlated, and we have one prototype per class positioned at their class-conditional mean. We would like to classify a training example from the north-west corner of the cluster of class 2. Recall that an example is assigned to the class of the closest prototype which highly depends on the employed metric. In Fig. 2b we visualize the non-adaptive Euclidean metric of GLVQ by its equi-distance lines which are concentric circles. Obviously, this example is wrongly classified. For GRLVQ, the employed metric can adapt to the dataset, but its equi-distance lines are restricted to axis-aligned ellipsoids. We show a possible instantiation of the equi-distance lines in 2c, but conclude that we are still not able to adapt the metric such that this example is correctly classified. In GMLVQ, the adaptive metric is extended such that we can construct rotated ellipsoidal equi-distance lines. The instantiation of equi-distance lines illustrated in Fig. 2d coincides with the shape of the class-specific clusters, and the considered data point is now correctly classified.

A final and important note is that the resulting classifiers of GRLVQ and GMLVQ are still interpretable since the relevance profiles correspond to the relative importance of the features for the classification task. Hammer et al. [19, 31] showed that the attractive worst-case generalization bounds of GLVQ can be transferred to the variants of GLVQ that incorporate metric learning.

2.2 FORMALIZATION

This section provides the preliminaries for the formal treatment of training and classification in GMLVQ. Especially, we make mathematically precise what we mean by training data, prototypes and a quadratic distance measure.

2.2.1 *Data*

In a standard machine learning scenario, we study a set of input objects, which are represented by numerical feature vectors, that need

to be discriminated into C classes. Formally, we consider a training set of size P consisting of N-dimensional vectors

$$\chi = \left\{\vec{x}^{\mu} \in \mathbb{R}^N\right\}_{\mu=1}^{P} \quad \text{with labels} \quad S^{\mu} = S(\vec{x}^{\mu}) \in \{1, \dots C\}. \tag{1}$$

2.2.2 *Prototypes*

GMLVQ represent the classification problem in terms of prototype vectors which are located in the same N-dimensional input space as the data. Prototypes are labelled with a class and approximate clusters of data points belonging to the same class. At least one prototype per class needs to be specified, leading to the following definition

$$W = \left\{\vec{w}^{j} \in \mathbb{R}^N\right\}_{j=1}^{L} \quad \text{with labels} \quad \sigma^{j} = \sigma(\vec{w}^{j}) \in \{1, \dots C\} \tag{2}$$

where $L \geqslant C$. The number of prototypes is a hyper parameter of the model and needs to be carefully specified by means of a validation procedure to expect good performance. Too few prototypes may not represent the data structure sufficiently, while too many prototypes may cause over fitting which leads to poor generalization of the classifier.

2.2.3 *Distance measure*

GMLVQ is further parameterized by a general quadratic distance measure of the form

$$d^{\Lambda}(\vec{x}, \vec{w}) = (\vec{x} - \vec{w})^{T} \Lambda (\vec{x} - \vec{w}) \tag{3}$$

where $\Lambda \in S_{+}^{N}$ is a positive semi definite matrix. Note that this measure is not necessarily a valid metric, but a so-called pseudo-metric, since a positive *semi*-definite Λ implies that $d(\vec{x}, \vec{y}) = 0$ is possible for $\vec{x} \neq \vec{y}$. As mentioned in the introduction, it is very instructive to think of the capacity of this pseudo-metric by realizing that it can construct any ellipsoidal equi-distance lines.

We can avoid a complicated optimization constraint in the training phase by noting that any positive semi-definite matrix can be written as

$$\Lambda = \Omega^{T} \Omega \quad \text{with} \quad \Omega \in \mathbb{R}^{M \times N}. \tag{4}$$

This decomposition of Λ into its square root Ω is not uniquely determined as explained in section 3.2.2.2. Substituting this observation into Eq. 3 leads to a pseudo-metric that correspond to the squared Euclidean distance after a linear transformation Ω of the original feature space:

$$d^{\Omega}(\vec{x}, \vec{w}) = (\vec{x} - \vec{w})^{T} \Omega^{T} \Omega (\vec{x} - \vec{w}) = [\Omega(\vec{x} - \vec{w})]^{2}. \tag{5}$$

Note that this representation also gives us the possibility to learn a low-dimensional transformation i.e. a rectangular Ω matrix with $M < N$ [12].

2.2.4 *Cost function preliminaries*

In order to formulate the GMLVQ cost function we present here the preliminaries. The sets

$$W_+^\mu = \{\vec{w} \mid \vec{w} \in W \text{ and } \sigma(\vec{w}) = S(\vec{x}^\mu)\}, \tag{6}$$

$$W_-^\mu = \{\vec{w} \mid \vec{w} \in W \text{ and } \sigma(\vec{w}) \neq S(\vec{x}^\mu)\} \tag{7}$$

contain the prototypes that have the same or different class as the example $\vec{x}^\mu$, respectively. We use the following indicator functions

$$\begin{aligned} \Psi^+(\vec{x}^\mu, \vec{w}) &= \begin{cases} +1 & \text{if } \vec{w} = \arg\min_{\vec{w}_i \in W_+^\mu} d^\Omega(\vec{x}^\mu, \vec{w}_i) \\ 0 & \text{else} \end{cases} \\ \Psi^-(\vec{x}^\mu, \vec{w}) &= \begin{cases} +1 & \text{if } \vec{w} = \arg\min_{\vec{w}_i \in W_-^\mu} d^\Omega(\vec{x}^\mu, \vec{w}_i) \\ 0 & \text{else} \end{cases} \end{aligned} \tag{8}$$

to identify the closest correct and closest wrong prototype, respectively. We use the abbreviations $\Psi_j^{\mu+}$ and $\Psi_j^{\mu-}$ to denote $\Psi^\pm(\vec{x}^\mu, \vec{w}_j)$, respectively. The indicator functions are used to determine the distance to the closest correct and closest wrong prototype

$$d_+^\mu = \sum_j \Psi_j^{\mu+} d^\Omega(\vec{x}^\mu, \vec{w}_j), \tag{9}$$

$$d_-^\mu = \sum_j \Psi_j^{\mu-} d^\Omega(\vec{x}^\mu, \vec{w}_j), \tag{10}$$

respectively. In the definition of the cost function we will use the abbreviations d_+^μ and d_-^μ to increase readability.

2.3 TRAINING

This section derives a training procedure for GMLVQ. We present the GMLVQ cost function, and show that an extra constraint on the norm of the matrix Ω is necessary. We formulate training as minimization of the empirical cost function, and present a projected Stochastic Gradient Descent method as optimization algorithm.

2.3.1 *Cost function*

Training in GMLVQ is guided by the following cost function

$$f(W, \Omega) = \sum_\mu \varphi(e^\mu) \quad \text{with} \quad e^\mu = \frac{d_+^\mu - d_-^\mu}{d_+^\mu + d_-^\mu} \tag{11}$$

where d_+^μ and d_-^μ refer to the distance of the closest correct and closest wrong prototype, respectively. The numerator of e^μ is related to the so-called hypothesis margin, and minimization of this term positively influences the generalization performance [19]. The denominator was introduced by Sato and Yamada [28] and has several effects. First, it restricts e^μ to the interval $[-1, 1]$ and therefore bounds the cost function from below[1]. Second, as argued in [28], it implicitly increases the force to the closest correct prototype, and therefore makes the prototypes more representatives of the data. Third, it provides invariance to the scale of distances i.e. we can multiply all distances by a scalar c without affecting e^μ as shown in 14. In section 3.1.1 we argue that this latter property is the main reason for GMLVQ to select low-rank transformation matrices.

Obviously, the term is negative for a correctly classified example, since d_+^μ must be smaller than d_-^μ. Hence, minimizing the cost function aims at minimization of the classification error and maximization of margins at the same time. A handle to balance the trade-off between the two terms is provided by the monotonically increasing scaling function φ. A popular choice is a logistic function of the form

$$\varphi(z) = \frac{1}{1 + \exp(-\gamma z)} \tag{12}$$

Here, γ controls the steepness of the sigmoid and the larger γ the more we approximate the $0-1$ loss function that directly corresponds to the classification error. On the other hand, for small γ or if we choose the identity function $\varphi(z) = z$ as scaling function, we prefer large margins over classification performance.

2.3.2 *Matrix constraint*

The adaptive measure is a valid pseudo-metric if and only if Λ is positive semi-definite. Therefore, in order to optimize directly over Λ we should add a constraint to the optimization problem to ensure that $\Lambda \succeq 0$ is positive semi-definite. We can avoid such a complicated optimization constraint by optimizing over Ω based on the decomposition in Eq. 35, since $\Omega^T\Omega$ is guaranteed to be positive semi-definite. Nevertheless, an extra constraint on the norm of Ω is needed to obtain a unique Ω. We show in the following that Ω can be multiplied with a scalar $c \neq 0$ without changing the cost function value. Let $\Omega' = c\Omega$, then the new distance measure reads as

$$\begin{aligned} d^{\Omega'}(\vec{x}, \vec{w}) &= (\vec{x} - \vec{w})^\top (c\Omega)^\top (c\Omega)(\vec{x} - \vec{w}) \\ &= [c\Omega(\vec{x} - \vec{w})]^2 \\ &= c^2 d^\Omega(\vec{x}, \vec{w}). \end{aligned} \tag{13}$$

1 It prevents that one prototype is pushed towards ∞ such that its hypothesis margin is $-\infty$.

Note that the closest correct and closest wrong prototype are not affected since each distance is multiplied with the same constant c^2. By plugging the new distance measure into the cost function of Eq. 11, we see that c^2 cancels out:

$$f(W, \Omega') = \sum_{\mu=1}^{P} \varphi \left[\frac{c^2 d_+^\mu - c^2 d_-^\mu}{c^2 d_+^\mu + c^2 d_-^\mu} \right] = \sum_{\mu=1}^{P} \varphi \left[\frac{d_+^\mu - d_-^\mu}{d_-^\mu + d_-^\mu} \right] = f(W, \Omega). \tag{14}$$

This implies that the cost function is invariant to the norm of Ω, and thus extra restrictions on Ω should be considered to obtain a unique solution.

Schneider et al. proposed to fix the sum of the diagonal values of $\mathrm{Tr}(\Lambda) = \sum_i \Lambda_{ii} = 1$, which also coincides with the sum of eigenvalues. More importantly, it follows that we restrict the Frobenius norm of Ω, since $\|\Omega\|_F = \mathrm{Tr}(\Omega^T \Omega) = \mathrm{Tr}(\Lambda) = 1$. This constraint singles out a unique norm, and thus solves the problem that the cost function is invariant to the Frobenius norm of Ω.

2.3.3 *Optimization problem*

We are now at the point that we can formulate training as the following optimization problem

$$\begin{aligned} &\underset{W,\Omega}{\text{minimize}} && f(W, \Omega) \\ &\text{subject to} && h(\Omega) = \|\Omega\|_F - 1 = 0 \end{aligned} \quad . \tag{15}$$

Here, the objective function is non-convex, but the constraint function is convex. It is very instructive to remark that the set of points that satisfy the constraint, the so-called feasible set, is a unit sphere. This follows when we rewrite the constraint function in element-wise form $\|\Omega\|_F = \sum_{ij} \Omega_{ij}^2 = 1$. Fig. 3 shows a 3D visualization of the constraint. The blue line correspond to similar Ω solutions of different frobenius norm. For the optimal Ω, the optimization landscape looks like a valley along the blue line. This latter fact will be important to derive the stationarity conditions for the transformation matrix in section 3.1.1. As a final note we mention that above optimization problem is a non-convex problem, and thus the best we can hope for is a solver that converges to a local minimum.

2.3.4 *Projected Stochastic Gradient Descent*

Traditionally, LVQ variants are optimized by Stochastic Gradient Descent (SGD). Schneider et al. follows this tradition, and proposed a projected SGD method to solve the optimization problem of Eq. 87. SGD iteratively updates the parameters in the direction of the negative gradient with respect to a *single* randomized training example.

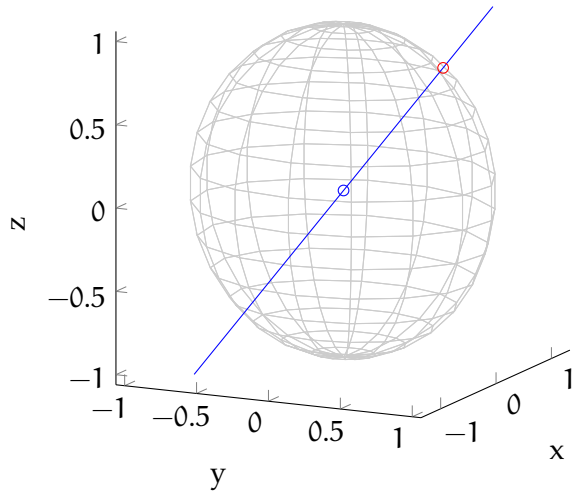


Figure 3: A 3D visualization of the constraint on the norm of Ω. The red spot marks an arbitrary Ω that lies on the unit sphere. Every other point that is on the line between the red spot and the origin corresponds to the same cost function value (except at the origin itself). The point where the blue line hits the other side of the sphere corresponds to another square root of Λ.

After each update step, the parameters are projected onto the feasible set.

We thoroughly discuss the stochastic gradient descent method for GMLVQ in section 4.1.1, and summarize the training procedure in Algorithm 1. Note that we do not include the projection step since it turns out to be possible[2]) to optimize without constraints (and then project in one step to the feasible set after training). In contrast, the projected SGD method of Schneider et al. includes a normalization of the Ω elements $\Omega_{ij} = \frac{\Omega_{ij}}{\sqrt{\sum_{kl} \Omega_{kl}^2}}$ after each stochastic gradient step. Convergence proofs can be derived for such projected SGD method under mild assumptions [5, 8, 9], although the rate of convergence is worse than plain SGD[3].

2 See section 3.3 for more details.

3 The standard convergence proof for stochastic gradient descent, in which expected cost function value decreases at each step, could be easily extended to our constrained optimization problem since the projection step does not affect the cost function value. It is not immediately clear whether this also implies that the rate of convergence is similar to ordinary SGD.

2.4 CLASSIFICATION

Classification is performed by a nearest prototype scheme. A data point $\vec{\xi}$ is assigned to the class of the closest prototype with respect to the pseudo-metric d^{Λ}. We can formulate this by

$$\vec{\xi} \leftarrow \sigma(w_i) \quad \text{with} \quad w_i = \underset{\vec{w}_j \in W}{\arg\min}\, d^{\Lambda}(\vec{\xi}, \vec{w}_j) \tag{16}$$

where ties can be broken arbitrarily. The set of prototypes W and the pseudo-metric d^{Λ} partition the input space. For each prototype w_i we have a so-called receptive field R_i that defines the region in feature space for which w_i is the closest prototype:

$$R_i = \{x \in \mathbb{R}^N \mid d^{\Lambda}(x, w_i) < d^{\Lambda}(x, w_j),\ \forall j \neq i\}. \tag{17}$$

3 STATIONARITY AND UNIQUENESS

This chapter investigates the stationarity conditions of the prototypes and the transformation matrix. For LVQ1, the stationarity conditions of the relevance matrix are already worked out in [3]. The authors show that for gradient based optimization, the relevance matrix has a tendency to become low rank. The results can, however, not be directly transferred to GMLVQ. Therefore, we rely on another approach: we phrase GMLVQ as an optimization problem and then investigate the first-order necessary conditions for optimality. One major advantage is that the results are not restricted to gradient based solvers, but are generally applicable to any type of solver, as long as it converges to an optimal solution. Moreover, we do not only investigate the stationarity conditions of the relevance matrix, but also of the prototypes.

We present the stationarity conditions in section 3.1. Key findings are that prototypes can be formulated as a linear combination of the data points, and that the transformation matrix has a tendency to become low rank. The results imply that the optimal prototypes and transformation matrix have many degrees of freedom, and that additional constraints must be formulated to obtain a unique and interpretable solution. We discuss and illustrate these ambiguities in section 3.2. Finally, section 3.3 incorporates the constraints in a new optimization problem, and proves that one constraint is automatically satisfied by gradient based optimization if the prototypes and transformation matrix are initialized in a particular way.

3.1 STATIONARITY CONDITIONS

The formulation of GMLVQ as an optimization problem allows to investigate the first-order necessary conditions for optimality. For constrained optimization problems these conditions are known as the KKT-conditions[10, 26]. The optimality conditions require that the cost function and its constraints are continuously differentiable. It is shown in the appendix of [18] that the cost function of GMLVQ fulfils this requirement, and one can easily see that the constraint is differentiable everywhere[1].

1 A quadratic function is differentiable, and a sum of differentiable function is differentiable.

3.1.1 *Stationarity of the matrix*

We first investigate the optimality conditions for the transformation matrix. The optimization problem of GMLVQ, as defined in Eq. 87, has an *equality* constraint, and the KKT conditions therefore require that the gradient of the cost function is parallel to the gradient of the constraint:

$$\frac{\partial f(W,\Omega)}{\partial \Omega} = \lambda \frac{\partial h(\Omega)}{\partial \Omega}, \tag{18}$$

where λ is a Lagrange multiplier. The full gradient of the cost function with respect to Ω is given by

$$\frac{\partial f(W,\Omega)}{\partial \Omega} = \Omega\Gamma \quad \text{with} \quad \Gamma = \sum_{\mu=1}^{P} \sum_{j} \chi_j^{\mu} \left(\vec{x}^{\mu} - \vec{w}^{j}\right) \left(\vec{x}^{\mu} - \vec{w}^{j}\right)^{\top} . \tag{19}$$

The pseudo-covariance matrix Γ collects covariances from the data points $\vec{x}^{\mu}$ to the prototypes $\vec{w}_j$. The exact contribution depends on the complex weighting factor χ_j^{μ} which is only non-zero for the closest correct and closest incorrect prototype. The contribution is negative for the closest incorrect prototype, and thus Γ is not necessarily positive semi-definite. We refer the reader to Appendix A.1.1 for an exact derivation of the gradient and its weighting factors. The gradient of the constraint is derived in Appendix A.1.2 and reads as

$$\frac{\partial h(\Omega)}{\partial \Omega} = 2\Omega. \tag{20}$$

We plug the gradients terms into Eq. 18 and obtain the following stationarity condition for the transformation matrix

$$\Omega\Gamma = \lambda\, 2\, \Omega \quad \text{or equivalently} \quad \Gamma\Omega^{\top} = \lambda\, 2\, \Omega^{\top}. \tag{21}$$

By writing $\Omega^{\top}$ into column-wise form $[\vec{\omega}_1, \ldots, \vec{\omega}_N]$, we immediately recognize an eigenvalue problem

$$\Gamma\vec{\omega}_i = 2\lambda\vec{\omega}_i. \tag{22}$$

for each column ω_i. This implies that each column of $\Omega^{\top}$ is a vector in the eigenspace of Γ which corresponds to one particular eigenvalue 2λ. Now, let us first consider the non-degenerate case where Γ has unique, ordered eigenvalues

$$\gamma_1 < \gamma_2 < \cdots < \gamma_N \quad \text{with orthonormal}^2 \text{ eigenvectors} \quad \vec{g}_1,\ \vec{g}_2,\ \ldots,\ \vec{g}_N. \tag{23}$$

It already follows from the stationarity condition in Eq. 21 that the relevance matrix Λ has rank one, no matter which eigenvector of Γ is

2 due to the fact that Γ is real and symmetric

in the rows of Ω. In the following we claim, however, that the only stable eigenvector of Γ corresponds to i) eigenvalue zero and ii) the smallest eigenvalue.

Let us first argue that the corresponding eigenvalue must be zero. In section 2.3.2 we have shown that the cost function value does not change when we multiply Ω by a scalar $c \neq 0$. This suggests that removing the constraint from the optimization problem will not improve the solution i.e. cost function value. In Fig. 3 we illustrate the constraint function on the transformation matrix, and show a blue line that represents similar Ω solutions of different norm. Now, it is easily seen that the constraint selects a point along a valley, which implies that its gradient must be zero for an optimal solution. As a consequence the right hand side of Eq. 21 should also be zero, and therefore the Lagrange multiplier $\lambda = 0$ for $\Omega \neq 0$. This latter fact is in accordance with the interpretation of Lagrange multipliers. Namely, a zero Lagrange multiplier means that the constraint can be relaxed without affecting the cost function [26].

The other eigenvectors of Γ with corresponding non-zero eigenvalues are also stationary points of the lagrangian function. They are, however, not local minima of the cost function because the gradient is not zero for those solutions. Formally, we can prove this by showing that the bordered Hessian is not positive definite for these stationary points [26]. We omit the proof, however, and rely on the intuitive argument from above.

By now, it should be clear that the only stable eigenvector of Γ has eigenvalue zero. In the following we also claim that this stable eigenvector corresponds to the smallest eigenvalue. The first argument is given in the treatment of [3]. The power iteration method allows for a simple stability analysis that shows that the only stable stationary point is the eigenvector of Γ that corresponds to the smallest eigenvalue[3]. This is not obviously shown in our approach, and therefore we rely on an *intuitive* argument by decomposing Γ into

$$\Gamma = \Gamma^+ - \Gamma^- \tag{24}$$

$$\text{with } \Gamma^+ = \left[\sum_{\mu=1}^{P} \frac{4\varphi'(e^\mu) d_-^\mu}{(d_+^\mu + d_-^\mu)^2} (\vec{x}^\mu - \vec{w}^+)(\vec{x}^\mu - \vec{w}^+)^\top \right]$$

$$\Gamma^- = \left[\sum_{\mu=1}^{P} \frac{4\varphi'(e^\mu) d_+^\mu}{(d_+^\mu + d_-^\mu)^2} (\vec{x}^\mu - \vec{w}^-)(\vec{x}^\mu - \vec{w}^-)^\top \right] .$$

3 The approach can, strictly speaking, not be transferred to GMLVQ. The pseudo-covariance matrix Γ is not stationary: changing Ω will always change Γ. Hence, we can not apply a power iteration method.

Here, Γ^+ and Γ^- are positive semi-definite matrices[4] that collect the covariances from all data points to the closest correct and closest incorrect prototypes, respectively. Of course, we would like to pick the eigenvector of Γ^+ with the smallest eigenvalue such that the distance to the closest correct prototype is as small as possible. On the other hand, we would like to pick the largest eigenvalue of Γ^- in order to have large distances to the closest incorrect prototypes. However, the minus sign in front of Γ^- changes the sign of the eigenvalues which implies that we are again aiming for the eigenvector with the smallest eigenvalue. Although there is no clear relationship between the eigenvectors of $\Gamma^\pm$ and the eigenvectors of Γ, we should not be surprised that the best cost function value is obtained if we also pick the eigenvector of Γ with the smallest eigenvalue. In short, we have shown that the stable stationary eigenvector of Γ has eigenvalue zero which must be the smallest eigenvalue. Hence, Γ is positive semi-definite.

We have shown that the rows of the transformation matrix Ω contain the eigenvector $\vec{g}_1$ of Γ which corresponds to the eigenvalue 0. If we take into account the norm constraint $\|\Omega\|_F = 1$, we find a particularly simple solution

$$\Omega = \begin{pmatrix} \alpha_1 \vec{g}_1^\top \\ \alpha_2 \vec{g}_1^\top \\ \vdots \\ \alpha_N \vec{g}_1^\top \end{pmatrix} \quad \text{with} \quad \sum_i^N \alpha_i^2 = 1. \tag{25}$$

Note that this transformation matrix Ω is not unique because we have freedom in choosing the α-factors. It corresponds to the fact that a square root Ω of Λ is not uniquely determined as explained in section 3.2.2.2. The resulting Λ will nevertheless be unique

$$\Lambda = \Omega^\top \Omega = \sum_i^N \alpha_i^2 \vec{g}_1 \vec{g}_1^\top = \vec{g}_1 \vec{g}_1^\top . \tag{26}$$

Hence, the stationary Λ has rank one, and its column space is spanned by $\vec{g}_1$. Note that $\vec{g}_1$ is also the only eigenvector of Λ with a non-zero eigenvalue of one.

In above considerations we assumed a eigenvector $\vec{g}_1$ that correspond to a unique zero eigenvalue. In case of degenerate eigenvalues

$$\gamma_1 = \gamma_2 = \cdots = \gamma_n = 0 < \cdots < \gamma_N, \tag{27}$$

the rows of Ω can be arbitrary linear combinations of the vectors $\vec{g}_1, \vec{g}_2, \ldots, \vec{g}_n$. Then, the rank of Λ is $1 \leqslant \operatorname{rank}(\Lambda) \leqslant n$. Note that it is

4 The scalar $\frac{4\varphi'(e^\mu) d_-^\mu}{(d_+^\mu + d_-^\mu)^2} \geqslant 0$ because the distances $d_\pm^\mu \geqslant 0$ and $\varphi'(e^\mu) \geqslant 0$ because φ is monotonically increasing. The matrix $(\vec{x}^\mu - \vec{w}^+)(\vec{x}^\mu - \vec{w}^+)^\top$ is positive definite by definition.

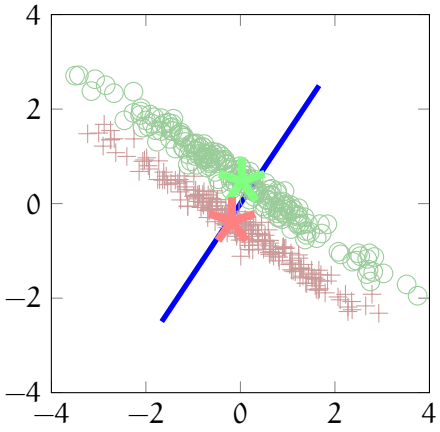


Figure 4: An example of the stationary Λ for a two dimensional dataset with two classes. The distance is measured along a single direction which is rendered as a blue line.

still possible to obtain a full rank relevance matrix if and only if we have N degenerate zero eigenvalues.

We conclude from the derived stationarity conditions that the optimal relevance Λ matrix is completely determined by pseudo-covariance Γ. Note, however, that the stationary pseudo-covariance Γ in turn depends on the outcome stationary relevance matrix. Hence, Γ and Λ are highly interconnected and it is not possible to predict pseudo covariance (or relevance matrix) from the data set alone. The dynamics of GMLVQ are thus not immediately clear from the presented analysis.

In the following argument we show, however, that GMLVQ has a tendency to select a low-dimensional Λ. Imagine we add an extra feature to the existing data set that consist of white Gaussian noise with variance σ^2. For all classes we draw from the same distribution, and therefore we can not expect discriminative power between the classes in this dimension. If we consider one prototype per class, then the value of the extra dimension is zero (= mean of the noise) for all prototypes. Therefore, we expect that the Euclidean distance between a data point and a prototype grows by σ^2. This include the distances between a data point and the correct closest and closest incorrect prototype, hence the cost function reads as

$$\varphi\left(\frac{(d_+^\mu+\sigma^2)-(d_-^\mu+\sigma^2)}{(d_+^\mu+\sigma^2)+(d_-^\mu+\sigma^2)}\right)=\varphi\left(\frac{d_+^\mu-d_-^\mu}{d_+^\mu+d_-^\mu+2\sigma^2}\right), \tag{28}$$

where $d_\pm^\mu$ are the distances to the closest correct and closest incorrect prototype in the dataset without the extra dimension. We notice that σ^2 cancels in the numerator, but not in the denominator. For a correctly classified example (negative $\frac{d_+^\mu-d_-^\mu}{d_+^\mu+d_-^\mu}$) the cost function value thus deteriorates. This simple example demonstrates that the GMLVQ cost function penalizes a dimension in which there is no discrimina-

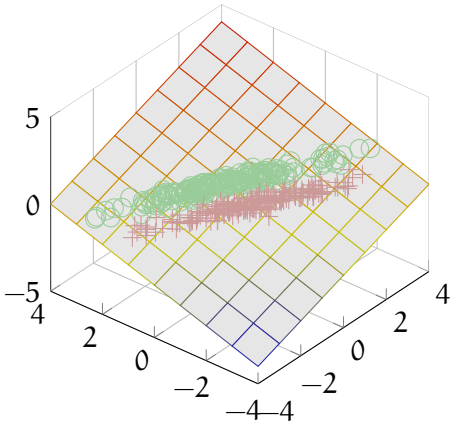


Figure 5: A three-dimensional data set that is embedded in a two-dimensional subspace as illustrated by a grey plane. The orthogonal direction to this plane is the null space of the data. The optimality conditions require that an optimal prototype lies on this grey plane.

tive power. We can generalize this example to a linear combination of dimensions in which there is no discriminative power. The main message is that it is beneficial for the GMLVQ cost function to cut off these non-discriminative directions, and therefore we expect a low-rank Λ, in general. Note that this extra dimension argument does not have a negative impact on an LVQ2.1 related cost function in which the denominator is left out. Therefore, we expect that incorporating quadratic metric learning in LVQ2.1 would, in general, not result in a low rank relevance matrix.

We conclude this section with an illustrative example of the stationary relevance matrix Λ for a simple two class dataset with highly correlated features as shown in Fig. 4. The stationary Λ has rank one and the distance is only measured along a single vector $\vec{g}_1$. The direction of this line is the only non-zero eigenvector of relevance matrix Λ and at the same time the eigenvector of pseudo-covariance Γ with eigenvalue zero.

3.1.2 *Stationarity of the prototypes*

This section presents the stationarity condition of the prototypes. We derive the first-order necessary conditions for optimality of the prototypes based on the optimization problem defined in Eq. 87. In contrast to the transformation matrix Ω, there are no constraints placed on the prototypes. Therefore the optimality conditions require only that the gradient of the cost function with respect to a single prototype $\vec{w}^k$ is zero. A detailed derivation of the gradient of the prototypes can be found in Appendix A.1.1, as well as a derivation of the

complex weighting factors χ_j^μ that depend on the role of the prototype. Here, we work out the gradient term to

$$\frac{\partial f(W,\Omega)}{\partial \vec{w}^k} = \sum_{\mu=1}^{P} \chi_k^\mu \Lambda \left(\vec{x}^\mu - \vec{w}^k\right) = 0 \tag{29}$$

$$= \Lambda \sum_{\mu=1}^{P} \chi_k^\mu \left(\vec{x}^\mu - \vec{w}^k\right) = 0 \tag{30}$$

$$= \Lambda \left(\sum_{\mu=1}^{P} \chi_k^\mu \vec{x}^\mu - \left[\sum_{\mu=1}^{P} \chi_k^\mu\right] \vec{w}^k \right) = 0, \tag{31}$$

where the last equation can be rewritten to the stationarity condition:

$$\Lambda \left(\vec{w}_{LC}^k - \vec{w}^k\right) = 0 \quad \text{with} \quad \vec{w}_{LC}^k = \frac{\sum_{\mu=1}^{P} \chi_k^\mu \vec{x}^\mu}{\sum_{\mu=1}^{P} \chi_k^\mu}. \tag{32}$$

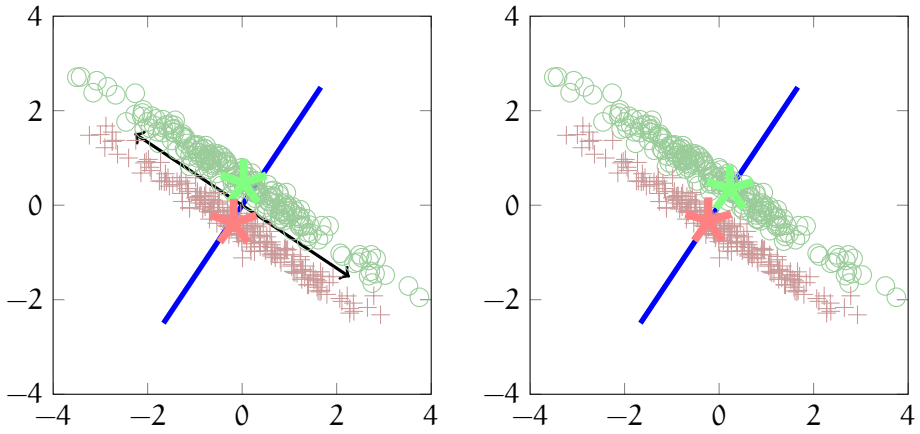
Now, it immediately follows that this equation is satisfied if $\vec{w}^k = \vec{w}_{LC}^k$. Here, $\vec{w}_{LC}^k$ is a prototype that is formulated as a linear combination of the data points, and thus it is in the span of the data. We give a concrete example of this result by considering the 3-D data set shown in Fig. 5. The 3 dimensional data, is equivalent to the 2-D data set shown in Fig. 4, but it is embedded in a two-dimensional subspace, as illustrated by a grey plane. Now, the prototype $\vec{w}_{LC}^k$ must lie on the plane. In the next section we show that $\vec{w}_{LC}^k$ is not the only stationary solution because the relevance matrix Λ has low rank. The singular Λ also implies that stationary prototypes are not necessarily in the span of the data if there is overlap between null space of the relevance matrix Λ and the null space of the data.

3.2 UNIQUENESS

This section summarizes the four ambiguities that arise in GMLVQ. We show in section 3.2.1 that contributions of the null space of Λ can be added to the prototypes. Section 3.2.2 presents the three types ambiguities of the transformation matrix: the norm of the transformation matrix Ω, the squareroot Ω of Λ, and underdetermined linear transformation Ω. For all ambiguities we present a projection to a *unique* and *interpretable* solution.

3.2.1 *Uniqueness of the prototypes*

In section 3.1.1 we have shown that the stationary relevance matrix Λ has low rank. In this section we show that a singular relevance matrix Λ implies that the prototypes are not unique. Let us first recall Eq. 32 where we presented the stationarity condition of a prototype $\vec{w}^k$. In



(a) Prototypes can be moved in the direction of the black line

(b) The null space of Λ is projected out of the prototypes

Figure 6: Illustrative example of the ambiguity of the prototypes. The single relevance vector is rendered as a blue line in a) and b). In a) we show that the prototypes can be moved in the direction of the null space of Λ which is rendered as a black line. In b) we show how to obtain a unique set of prototypes by projecting out the null space of Λ.

the previous section we have worked out the straightforward solution $\vec{w}^k = \vec{w}^k_{LC}$. Here, we show that it also possible to have stationary prototypes if $\vec{w}^k \neq \vec{w}^k_{LC}$. In that case, the difference vector $z = \vec{w}^k_{LC} - \vec{w}^k$ has to satisfy $\Lambda z = 0$. In other words, only contributions from the null space[5] of Λ can be added to the prototypes

$$\vec{w}^k = \vec{w}^k_{LC} + z \quad \text{with} \quad z \in N(\Lambda). \tag{33}$$

Note that this null space contributions leave the distances $d^{\Lambda}(\vec{x}, \vec{w})$ unchanged for every data point $\vec{x}$, hence the resulting classifier is identical to the one obtained for $\vec{w}^k = \vec{w}^k_{LC}$. We illustrate the implications of this finding in Fig. 6a. The one-dimensional null space of Λ is rendered as a black line, and is orthogonal to the blue line that represents the column space of Λ. Now, the prototypes can be moved in the direction of the black line without changing the classifier.

The ambiguity of the prototypes suggest that we have to be careful with the interpretation of the prototypes as class representatives. Obviously, the prototypes are not representatives of the classes in the *original* feature space. In Fig. 6a the prototypes could be moved very far from the data along the black line, while the classifier would still be optimal. Hence, we have to keep in mind that the prototypes are only representative in the space that is obtained after a linear transformation Ω of the original feature space. However, we have shown that this space is often low-dimensional, and one way to ensure that

5 Note that the null space of the relevance matrix is equal to the null space of transformation matrix, that is, $N(\Lambda) = N(\Omega)$.

the prototypes live in this very same space[6] is by removing null space contributions of Λ. We obtain a unique prototype

$$\vec{\tilde{w}}^k = \Lambda^+(\Lambda\vec{w}^k) \quad \text{where} \quad \Lambda^+ \quad \text{is the pseudo-inverse,} \tag{34}$$

that is in the column space of relevance matrix[7] $\vec{\tilde{w}}^k \in C(\Lambda)$. The column space projection results in a prototype that has smallest l^2 norm among all optimal prototypes. Note that this is not the only way to obtain a set of unique prototypes. For instance, we could also restrict to a prototype $\vec{w}^k_{LC}$ that is a particular linear combination of the data points.

We conclude this section with an illustration of prototypes that are projected in the column space of Λ. In Fig. 6b we demonstrate that unique prototypes are obtained by projection of an arbitrary optimal prototype on the blue line.

3.2.2 *Uniqueness of the transformation matrix*

In this section we show that even more ambiguities play a role in the transformation matrix Ω.

3.2.2.1 *Norm of the transformation matrix*

In the original GMLVQ paper [31] the authors propose to fix $\|\Omega\|_F = 1$ in order to prevent degenerate Ω matrices i.e. a Ω matrix that approaches the zero matrix or escapes to infinity. We point out, however, that the cost function does not have a tendency to approach the degenerate solutions, in general. We also include the norm constraint $\|\Omega\|_F = 1$ in the optimization problem, but to single out a unique norm Ω since we have shown in section 2.3.2 that it does not influence the cost function. In other words, we could leave out the constraint and then project (after optimization) to a unique norm Ω solution by dividing all elements of Ω by $\sqrt{\|\Omega\|_F}$. In Fig. 3 this relates to a projection of an Ω solution on the unit sphere.

3.2.2.2 *Square root Ω of Λ*

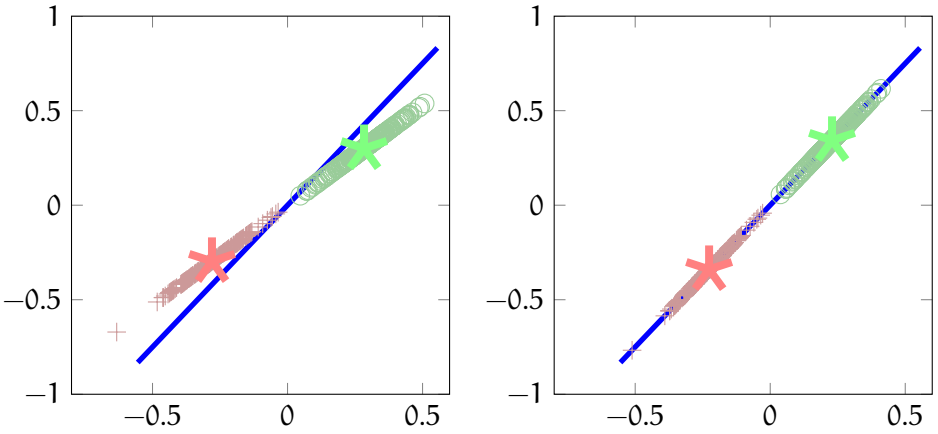
In the distance measure, we have used the fact that any positive semi-definite matrix can be written as

$$\Lambda = \Omega^T\Omega \quad \text{with} \quad \Omega \in \mathbb{R}^{N\times N}. \tag{35}$$

This decomposition of Λ into its square root Ω is not uniquely determined. We can multiply Ω by an arbitrary orthogonal (less formally,

6 apart from rotations and reflections of the low-dimensional space itself

7 Column space of the relevance matrix is equal to the row space of the transformation matrix, $C(\Lambda) = C(\Omega^T)$



(a) Arbritary square root Ω (b) Unique square root $\hat{\Omega}$

Figure 7: The blue line illustrates the only non-zero eigenvector of Λ. In a) we show an arbitrary transformation Ω of the data and the prototypes. Remark that such a transformation is rotated with respect to the blue line. In b) we show a unique transformation $\hat{\Omega}$ that realizes a projection of the data points and prototypes on the blue line.

a rotation/reflection) matrix R, so that $R^\top R = I$, without changing Λ. This immediately follows when we let $\Omega' = R\Omega$, since

$$\begin{aligned}\Omega'^{\top}\Omega' &= (R\Omega)^\top(R\Omega)\\ &= \Omega^\top R^\top R\Omega\\ &= \Omega^\top\Omega. \end{aligned} \tag{36}$$

To link this technical observation to the GMLVQ classifier, recall that the quadratic pseudo-metric d^Ω measures the squared Euclidean distance in a linear transformation Ω of the original feature space. We can, however, always rotate the coordinate system (or permute coordinate axes) in this transformed space without changing the distances. This is exactly what multiplication by an orthogonal matrix R in Eq. 36 realizes. Hence, the coordinates of the linear mapping are determined by Λ, but there is some freedom in choosing the coordinate system.

It is, however, possible to obtain a unique square root $\hat{\Omega}$ if we agree on the coordinate system (set of basis vectors) we use. Because Λ is positive semi-definite, it is possible to do an eigendecomposition

$$\Lambda = VDV^\top, \tag{37}$$

where the columns of V contain the orthonormal eigenvectors of Λ , and the diagonal entries of D contain the eigenvalues in *ascending* order. Now, we define a unique transformation matrix

$$\hat{\Omega} = VD^{0.5}V^\top, \tag{38}$$

that is the only square root that is positive semi-definite i.e. $\hat{\Omega} \succeq 0$. Note that we fix the coordinate system of the transformed space by taking the eigenvectors of Λ as a orthogonal basis. Of course, other coordinate systems are possible. For instance we could choose $VD^{0.5}$ such that we arrive at the standard coordinate system (with unit vectors as basis). Note that this choice requires, in contrast to $\hat{\Omega}$, to fix the sign of the eigenvectors in order to have a unique transformation matrix. Therefore, we feel that $\hat{\Omega}$ is a more natural solution, and illustrate an example transformation in Fig. 7. The transformation $\hat{\Omega}$ realizes a projection of the data on the blue line, while an arbitrary transformation Ω often realizes a rotation of the data and prototypes with respect to this blue line.

3.2.2.3 *Underdetermined linear transformation* Ω

The distance measure implicitly realizes a linear mapping Ω of the data X and the prototypes W, as pointed out in Eq. 3. In the following we assume, without loss of generality, that prototypes are in the span of the data, and we thus consider a linear mapping of the data points:

$$\Omega\vec{x}^{\mu} = \vec{y}^{\mu}, \quad \text{for all} \quad \mu = 1, \text{dots}, P. \tag{39}$$

or in matrix form

$$\Omega X = Y, \tag{40}$$

where $X = [\vec{x}^1, \ldots, \vec{x}^P]$ and $Y \in \mathbb{R}^{N \times P}$ denote the data and transformed data matrix, respectively. Such a mapping is uniquely determined if and only if the rank of the data matrix X is N. This requirement is, in general, not satisfied because $\text{rank}(X) < N$ in the following cases. If i) the number of features simply exceeds the number of data points or ii) the features in the data set are highly correlated such that the data set is embedded in a low-dimensional subspace (e.g. see Fig. 5). For such data sets it is possible to add contributions z from the left null space of data, i.e. $zX = 0$, to the rows ω_i of Ω without changing the linear transformation. Hence, many equivalent linear transformations exist, and the degree of freedom is given by the dimension of null space of X. Note that, in contrast to the previous section, the equivalent linear transformations result in a different relevance matrix Λ.

Strickert et al. [38] demonstrate that it is also misleading to directly interpret the relevance matrix Λ in the setting of an under determined linear mapping. The interpretability can only be guaranteed if we remove null space contributions from the transformation matrix Ω. The authors propose an explicit regularization scheme, while we use the well-known Moore-Penrose pseudo inverse to obtain a unique linear transformation

$$\tilde{\Omega} = (\Omega X)X^{+} \quad \text{with} \quad X^{+} \quad \text{the pseudo-inverse.} \tag{41}$$

The regularization enforces the rows $\tilde{\omega}_i$ of $\tilde{\Omega}$ in the span of the data, that is, $\tilde{\omega}_i \in C(X)$. Note that $\tilde{\Omega}$ also corresponds to a least Frobenius norm solution of the under determined linear mapping.

Finally, we remark that the regularization might improve the generalization performance of the classifier. Consider Fig. 5 where we have a data set that is embedded in a low-dimensional space, and furthermore assume that we have a learned a linear mapping for this particular data set. Imagine that the classifier encounters a new data point that does not lie on the grey plane. Then, the unique transformation matrix $\tilde{\Omega}$ maps this data point and the projection of this data point on the grey plane to exactly the same point in the transformed space. In other words, differences along the null space of the original data set are not taken into account. All other transformations map the two points to (very) different points in the transformed space, while there is no statistical justification from the training set for these differences.

3.3 NEW OPTIMIZATION PROBLEM

In section 3.2 we have explained what type of ambiguities can arise in the GMLVQ classifier, and we also presented projections to obtain a unique set of prototypes and a unique transformation matrix. Here, we incorporate these constraints directly in the optimization problem:

$$\begin{aligned} \underset{W,\Omega}{\text{minimize}} \quad & f(W,\Omega) = \sum_{\mu=1}^{P} \varphi\left[\frac{d_+^\mu - d_-^\mu}{d_-^\mu + d_-^\mu}\right] \\ \text{subject to} \quad & \|\Omega\|_F && = 1 \\ & \Omega && \succeq 0 \\ & \vec{w}^k \in C(\Lambda) && \forall k \\ & \omega_i \in C(X) && \forall i \end{aligned} \quad . \tag{42}$$

The first constraint $\|\Omega\|_F = 1$ requires the transformation matrix to lie on the unit sphere as explained in section 2.3.2. The second constraint singles out the only square root of the relevance matrix that is positive semi-definite. Note that the set of positive semi-definite matrices is convex [10]. The third constraint restrict a prototype $\vec{w}^k$ to the column space of Λ and the fourth constraint limits the rows ω_j to the span of the data. Both sets are affine subspaces and thus convex. Since an intersection of convex sets is convex, the feasible set of above optimization problem is convex.

It is possible to optimize such problem by a (stochastic) projected gradient descent method, in which each gradient step is followed by a projection of the parameters on the feasible set. Although convergence can be guaranteed for such procedure [9], the rate of convergence is often slow because the projection step deteriorates the

cost function value. Here, the projections do not affect the cost function value, and thus we expect similar convergence speed as plain (stochastic) gradient descent. It also opens up the possibility to optimize Eq. 42 without its four constraints, and then project the solution after training to the feasible set. We thereby avoid the computationally expensive projection steps at each iteration, which is especially attractive for stochastic gradient descent or other optimization methods that perform a large number of steps. In section 3.3.1 we show that we can actually eliminate the fourth constraint if clever initialize the transformation matrix and the prototypes.

Finally, we remark that the order of the four projections matters. We should first remove null space contributions of the data from Ω (the fourth constraint) since it changes Λ, while the other projections could be realized independent of each other.

3.3.1 *Elimination of fourth constraint*

In the following we prove that gradient update steps do not pickup irrelevant feature directions, and thus gradient-based optimization with specific initialization will satisfy the fourth constraint. That is to say, such approach automatically selects the least norm solution of the under determined linear mapping, and also ensures that prototypes are in the span of the data.

We prove the claim by induction on the gradient update steps. As basis step we initialize the transformation matrix Ω such that its rows do not contain contributions from the null space of the data, that is, $\omega_i \in C(X)$. We could for instance initialize the transformation matrix as the covariance matrix of the data, or as a random matrix and subsequently remove null space contribution of the data by $\tilde{\Omega} = (\Omega X)X^+$. Moreover, the prototypes need to be initialized within the span of the data, $\vec{w}^k \in C(X)$. It is rather mild assumption that is for instance satisfied if prototypes are initialized as the class conditional means.

As induction step we have to show that the gradient update keep prototypes and the rows of the transformation matrix in the span of the data. We first consider the prototype $\vec{w}_t^k$, at the current iteration, that is in the span of the data $C(X)$. Recall that the next prototype $\vec{w}_{t+1}^k$ is defined by the the following stochastic gradient update step:

$$\vec{w}_{t+1}^k = \vec{w}_t^k - \eta\, \chi_k^\mu\, \Omega^\top \Omega \left(\vec{x}^\mu - \vec{w}_t^k\right), \tag{43}$$

with learning rate η and weighting factor χ_k^μ. We first show that $\Omega^\top \Omega \left(\vec{x}^\mu - \vec{w}_t^k\right) \in C(X)$. Let $\vec{u} = \Omega \left(\vec{x}^\mu - \vec{w}_t^k\right)$, then it follows by definition of matrix multiplication that $\Omega^\top \vec{u} \in C(\Omega^\top)$. The column space of $\Omega^\top$ is equal to the row space of Ω, and thus $\Omega^\top \Omega \left(\vec{x}^\mu - \vec{w}_t^k\right)$ can be written as a linear combinations of the rows of Ω. Since the

rows $\omega_i \in C(X)$, we have $\Omega^\top \Omega \left(\vec{x}^\mu - \vec{w}^k\right) \in C(X)$. By definition of a linear subspace, multiplication of an element of the subspace by a scalar is in the linear subspace, and addition of two elements of the subspace is also in the subspace. Therefore, we remain in the span of the data if we multiply by scalars η and χ^μ_k, and add prototype $\vec{w}^k_t$, which was by assumption in $C(X)$.

Secondly, we consider the transformation matrix Ω_t and prove that the rows of Ω_{t+1} are also in $C(X)$. To this end, recall the gradient update for the transformation matrix:

$$\Omega_{t+1} = \Omega_t - \eta\, \Omega_t \sum_j \chi^\mu_j \left(\vec{x}^\mu - \vec{w}^j_t\right) \left(\vec{x}^\mu - \vec{w}^j_t\right)^\top \tag{44}$$

Let us first examine a single prototype contribution $\eta\, \Omega_t\, \chi^\mu_j \left(\vec{x}^\mu - \vec{w}^j_t\right) \left(\vec{x}^\mu - \vec{w}^j_t\right)^\top$. We define the difference vector $\vec{z} = \vec{x}^\mu - \vec{w}^j_t$, which is in $C(X)$ because $\vec{w}_t \in C(X)$. Furthermore, we ignore the scalar multiplication and thus we obtain $\Omega_t\, \vec{z}\vec{z}^\top$. The rows of this matrix can be written as a linear combination of $\vec{z}^\top$:

$$\Omega_t\, \vec{z}\vec{z}^\top = \begin{bmatrix} (\Omega_t \vec{z})_1\, \vec{z}^\top \\ \vdots \\ (\Omega_t \vec{z})_N\, \vec{z}^\top \end{bmatrix} \tag{45}$$

Since $\vec{z}$ is in the span of the data, it follows that the rows of $\Omega_t\, \vec{z}\vec{z}^\top \in C(X)$. The rows of single prototype contribution $\eta\, \chi^\mu_j\, \Omega_t\, \vec{z}\vec{z}^\top \in C(X)$ remains in the span of the data if we multiply by scalars η and χ^μ_j, or add rows of Ω_t, which were by assumption in the span of the data. The total gradient term is a sum of the single prototype contributions, which are all the span of the data, and thus remains in the span of the data by definition of a linear subspace.

Note that it is straightforward to extend the proof to full gradient terms, since it accumulates single-example contributions.

4 COMPARISON OF OPTIMIZATION METHODS

In previous chapters we have phrased the training procedure of GMLVQ as an optimization problem, and investigated the properties of its solution. However, we did no pay attention how to find the optimal prototypes and transformation matrix. Many optimization methods can be used (see e.g. Nocedal and Wright [26], Boyd and Vandenberghe [10] for an overview), and the goal of this chapter is to compare the performance of several algorithms on the optimization problem of GMLVQ.

It is very important to remark that the objective in machine learning is different from a standard optimization setting. Although we formulate the learning procedure as a minimization problem of the loss on training data, we really care about the loss on *test data* i.e. the generalization performance. Therefore, one should realize that state-of-the-art optimizers are not necessarily the best learning algorithms[7]. Bottou and Bousquet [6] provide an asymptotic analysis of batch and stochastic optimization methods in a large-scale machine learning setting, that is, the learning problem is constrained by computing time rather than the number of data points. They show that i) stochastic algorithms outperform batch algorithms despite being worse optimizers and ii) that introducing second-order information in stochastic gradient descent only wins us a constant factor (the condition number)[1]. Therefore, the best learning algorithms are between first- and second-order stochastic gradient descent, depending on the condition number[2] and the cost of taking curvature information into account. As a consequence, several methods were proposed that incorporate second-order information in stochastic gradient descent[4, 32, 23].

We emphasize that these theoretical results only hold in a large-scale setting; For small-scale learning problems is often beneficial to use sophisticated batch methods[24, 2]. Moreover, the second-order batch methods can be effectively parallelized on GPUs, and therefore remain competitive in large scale learning problems[25]. In short, the debate whether to use sophisticated batch optimization or simple stochastic gradient descent is ongoing, and depends on the learning problem at hand.

In section 3.3 of the previous chapter we have argued that it is possible to optimize the GMLVQ problem without constraints, and then project the solution after training to the feasible set. In this chapter we pursue this strategy and thus consider the learning problem

1 In contrast, introducing second-order information in batch methods results in exponential gain

2 The ratio of the largest to the smallest eigenvalue of the Hessian matrix.

as an unconstrained optimization problem. We present an comparative study of the performance of the following optimization methods on artificially generated GMLVQ problems. First, we evaluate a plain stochastic gradient descent (SGD) method. It is often preferred in the context of LVQ because it is simple to implement, fast and close in spirit to Kohonen's heuristic update rules. However, in order to obtain good performance, it is crucial to carefully tune the learning rates. This might be hard for inexperienced users, and thus plain SGD is not really appropriate for an off-the-shelf GMLVQ toolbox[3]. Second, we investigate adaptive learning rates for SGD as proposed by Schaul et al. [30], and which recently attracted much attention in the deep learning community. The approach is completely hyper-parameter free and estimates the learning rates by means of curvature estimation and the *variance of the gradient*. Third, we evaluate a (limited memory) BFGS method, which is regarded as one of the most successful quasi-Newton method. It was already suggested by [37] to use a BFGS method to optimize the GMLVQ cost function[4], but the authors never systematically compared the performance with stochastic gradient descent.

4.1 OPTIMIZATION METHODS

In the following subsections we review the used optimization methods.

4.1.1 *Stochastic Gradient Descent*

We start with applying stochastic gradient descent (SGD) to the optimization problem of GMLVQ. SGD iteratively updates the parameters in the direction of the negative gradient with respect to a *single* randomized training example. We therefore start by presenting the gradient terms of GMLVQ cost function as explicitly derived in Appendix A.1.

We obtain the gradient term with respect to a particular prototype $\vec{w}^k$ of a single example contribution

$$\frac{\partial \varphi(e^\mu)}{\partial \vec{w}^k} = \chi_k^\mu \Lambda \left(\vec{x}^\mu - \vec{w}^k\right) \tag{46}$$

where χ_k^μ is a scalar that depend on the actual role of the prototype:

CLOSEST CORRECT PROTOTYPE If $\vec{w}^k$ is the closest correct prototype to the example $\vec{x}^\mu$, than the term $\chi_k^\mu = -4\,\varphi'(e^\mu)\frac{d_-^\mu}{(d_+^\mu + d_-^\mu)^2}$ is negative. Hence a step in the direction of the negative gradient will push the prototype towards the considered example.

3 Nevertheless, we could include a grid search to find the optimal hyper-parameters.

4 The authors provides a toolbox that is available at http://mloss.org/software/view/323/

CLOSEST WRONG PROTOTYPE If $\vec{w}^k$ is the closest wrong prototype to the example $\vec{x}^\mu$, than the term $\chi_k^\mu = 4\,\varphi'(e^\mu)\frac{d_+^\mu}{(d_+^\mu + d_-^\mu)^2}$ is positive and a step in the direction of the negative gradient will push the prototype away from the considered example.

OTHERWISE If $\vec{w}^k$ is neither the closest correct nor the closest wrong, the term $\chi_k^\mu = 0$ and the prototype will not be updated.

Analogously, we obtain the derivative of e^μ with respect to the matrix Ω:

$$\frac{\partial e^\mu}{\partial \Omega} = \Omega \sum_j \chi_j^\mu \left(\vec{x}^\mu - \vec{w}^j\right)\left(\vec{x}^\mu - \vec{w}^j\right)^\top \tag{47}$$

where the weighting factors χ_j^μ are defined above.

In short, the Stochastic Gradient Descent update for a prototype reads as

$$\vec{w}^k = \vec{w}^k - \eta_W \chi_k^\mu \Lambda \left(\vec{x}^\mu - \vec{w}^k\right), \tag{48}$$

and the updates for the matrix as

$$\Omega = \Omega - \eta_\Omega\, \Omega \sum_j \chi_j^\mu \left(\vec{x}^\mu - \vec{w}^j\right)\left(\vec{x}^\mu - \vec{w}^j\right)^\top \tag{49}$$

where η_W and η_Ω are the learning rates for the prototypes and matrix, respectively. We summarize the SGD procedure for GMLVQ in Algorithm 1.

Convergence proofs can be derived for a SGD method under mild assumptions [5, 8], with a convergence rate of $\mathcal{O}(\frac{1}{t})$ at best for strongly convex functions [5]. The most important condition is that the learning rates must be square-summable, but not summable

$$\sum_t^\infty \eta_t^2 < \infty, \sum_t^\infty \eta_t = \infty. \tag{50}$$

The following frequently used learning rate schedule satisfies this condition

$$\eta(t) = \frac{\eta_{\text{start}}}{1 + t\Delta\eta} \tag{51}$$

where t denotes the current epoch, η_{start} the initial learning rate and $\Delta\eta$ the decaying factor. Furthermore, notice that we have introduced separate learning rates for the prototype and transformation matrix variables. This suggest that the curvature (second derivative) of both groups of variables are significantly different for most data sets.

5 For non-strongly convex function convergence rate is $\mathcal{O}(\frac{1}{\sqrt{t}})$

Algorithm 1 SGD procedure for GMLVQ

Require: Learning rates η_W, η_Ω
Require: nrEpochs
  for $i = 1 \rightarrow$ nrEpochs **do**
    for $\mu = 1 \rightarrow P$ **do**
      Determine closest correct and closest wrong prototype
      $\vec{w}_+ = \arg\min_{\vec{w}_i \in W^\mu_+} d^\Omega(\vec{x}^\mu, \vec{w}_i)$
      $\vec{w}_- = \arg\min_{\vec{w}_i \in W^\mu_+} d^\Omega(\vec{x}^\mu, \vec{w}_i)$

      Update prototypes
      $\vec{w}_+ = \vec{w}_+ + \eta_W 4\, \varphi'(e^\mu) \frac{d^\mu_-}{(d^\mu_+ + d^\mu_-)^2} \Lambda \left(\vec{x}^\mu - \vec{w}^k\right)$
      $\vec{w}_- = \vec{w}_- - \eta_W 4\, \varphi'(e^\mu) \frac{d^\mu_+}{(d^\mu_+ + d^\mu_-)^2} \Lambda \left(\vec{x}^\mu - \vec{w}^k\right)$

      Update matrix
      $\Omega = \Omega - \eta_\Omega\, \Omega\, 4\, \varphi'(e^\mu) \frac{d^\mu_-}{(d^\mu_+ + d^\mu_-)^2} \left(\vec{x}^\mu - \vec{w}_+\right) \left(\vec{x}^\mu - \vec{w}_+\right)^\top$
      $\Omega = \Omega + \eta_\Omega\, \Omega\, 4\, \varphi'(e^\mu) \frac{d^\mu_+}{(d^\mu_+ + d^\mu_-)^2} \left(\vec{x}^\mu - \vec{w}_-\right) \left(\vec{x}^\mu - \vec{w}_-\right)^\top$
    end for
  end for

4.1.2 *Adaptive learning rates for SGD*

This section presents adaptive learning rates for SGD as recently proposed by Schaul et al. [30]. In the following we consider a standard machine learning setting, in which samples $\vec{x}^\mu$ are independently drawn from an unknown distribution $\mathcal{P}$. For each sample, we associate a *per-sample loss* $e^\mu(\vec{\theta})$ that contributes to the expected loss:

$$f(\vec{\theta}) = E_{\mu \sim \mathcal{P}} \left[e^\mu(\vec{\theta}) \right], \tag{52}$$

where $\vec{\theta} = \mathbb{R}^d$ are adaptive model parameters, whose optimal value is denoted by $\vec{\theta}^* = \arg\min_{\vec{\theta}} f(\vec{\theta})$. Note that for GMLVQ the per-sample loss function is given by $e^\mu(\vec{\theta}) = \varphi(\frac{d^\mu_+ - d^\mu_-}{d^\mu_+ + d^\mu_-})$, with prototypes and transformation matrix concatenated into a model parameter vector $\vec{\theta} = \begin{bmatrix} W \\ \Omega \end{bmatrix}$.

In order to optimize the expected loss, we consider a stochastic gradient descent method with iterations of the form

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \eta_t \nabla e^\mu, \tag{53}$$

where $\nabla e^\mu = \left.\frac{\partial e^\mu(\vec{\theta})}{\partial \vec{\theta}}\right|_{\vec{\theta} = \vec{\theta}_t}$ is the sample gradient evaluated at $\vec{\theta}_t$, and η_t is the learning rate at iteration t.

4.1.2.1 *Quadratic approximation of loss function*

We assume that the per-sample loss function e^{μ} is smooth and can locally be approximated by a second-order Taylor expansion around $\vec{\theta}_0$:

$$\hat{e}^{\mu}(\vec{\theta}) = e^{\mu}(\vec{\theta}_0) + (\vec{\theta} - \vec{\theta}_0)^{\top} \nabla e^{\mu} + \frac{1}{2}(\vec{\theta} - \vec{\theta}_0)^{\top} H^{\mu} (\vec{\theta} - \vec{\theta}_0), \quad (54)$$

where $H^{\mu} = \frac{\partial^2 e^{\mu}(\vec{\theta})}{\partial^2 \vec{\theta}}\Big|_{\vec{\theta}=\vec{\theta}_0}$ is the positive semi-definite Hessian matrix evaluated at $\vec{\theta}_0$. We rewrite the quadratic approximation to:

$$\hat{e}^{\mu}(\vec{\theta}) = \underbrace{e^{\mu}(\vec{\theta}_0) + (2H^{\mu})^{-1}(\nabla e^{\mu})^2}_{=\text{constant}} + \frac{1}{2}(\vec{\theta} - \vec{\theta}^{\mu *})^{\top} H^{\mu} (\vec{\theta} - \vec{\theta}^{\mu *}), \quad (55)$$

where $\vec{\theta}^{\mu *} = \vec{\theta}_0 - (H^{\mu})^{-1} \nabla e^{\mu}$ denotes the minimum of the quadratic approximation of the loss function at $\vec{\theta}_0$. Note that the sample minimum is obtained after the well-known Newton step. For each sample we have a different minimum $\vec{\theta}^{\mu *}$, and the result of a batch newton step will be a complex weighted average of the sample minima. Further note that the first two-terms in above formula are constant, we will ignore them throughout the rest of this section since they do not influence the presented analysis. The minimum of the per-sample loss is thus zero.

In order to simplify the analysis, we assume that the Hessian is identical for each per sample loss. We obtain the following quadratic approximation of the loss:

$$\hat{e}^{\mu}(\vec{\theta}) = \frac{1}{2}(\vec{\theta} - \vec{c}^{\mu})^{\top} H^{*} (\vec{\theta} - \vec{c}^{\mu}), \quad (56)$$

where $\vec{c}^{\mu}$ is a multivariate random variable with mean $\vec{\theta}^{*}$ and covariance Σ. In other words, the sample minima are distributed according to some distribution, whose expected value is $\vec{\theta}^{*}$, the optimal batch newton step. Fig. 8 illustrates the setting in one dimension.

We further assume that the problem is separable, meaning that we implicitly assume that the Hessian and covariance matrix are diagonal i.e. $H = \text{diag}(h_1, \ldots, h_i, \ldots, h_d)$ and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_i, \ldots, \sigma_d)$. This makes it possible to treat each dimension i independently, and in the following we omit index i for brevity purposes. We thus obtain the one-dimensional problem:

$$\begin{aligned}
\hat{f}(\theta) &= E_{\mu \sim P}\left[\frac{1}{2} h (\theta - c^{\mu})^2\right] \\
&= E_{\mu \sim P}\left[\frac{1}{2} h (\theta^2 - 2\theta\, c^{\mu} + (c^{\mu})^2)\right] \\
&= \frac{1}{2} h (\theta^2 - 2\theta\, \theta^{*} + \theta^{*2} + \sigma^2) \\
&= \frac{1}{2} h ((\theta - \theta^{*})^2 + \sigma^2), \quad (57)
\end{aligned}$$

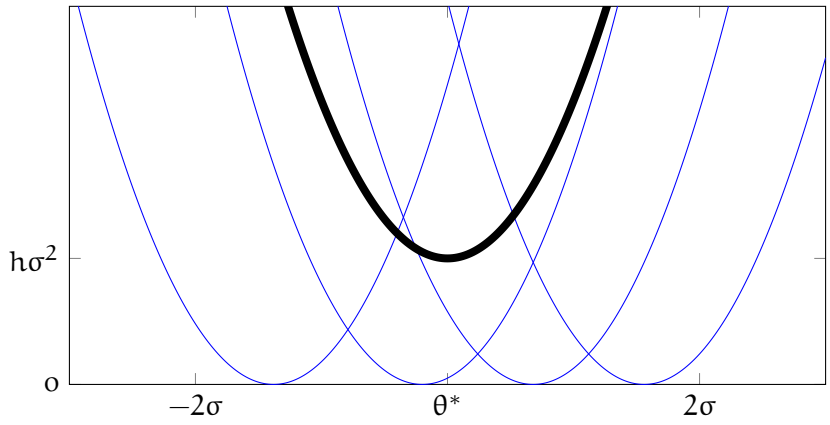


Figure 8: Idealized expected loss $\hat{f}_i(\vec{\theta})$ that is illustrated by a thick black line. The expected loss is an average of the quadratic sample losses $\hat{e}^\mu(\vec{\theta})$, which are shown as blue lines, and whose minima $\vec{\theta}_i^{\mu*}$ are distributed around $\vec{\theta}^*$ with variance σ_i. Note that the curvature is assumed to be identical for each sample.

where the second step follows from the fact that $E[c^2] = \mathrm{Var}(c) + E^2[c] = \sigma^2 + (\theta^*)^2$. The gradient with respect to a single example is given by

$$\nabla \hat{e}^\mu = h(\theta - c^\mu), \tag{58}$$

with expectation:

$$E[\nabla \hat{e}^\mu] = h(\theta - \theta^*), \tag{59}$$

and variance:

$$\mathrm{Var}(\nabla \hat{e}^\mu) = E[(\nabla \hat{e}^\mu)^2] - E^2[\nabla \hat{e}^\mu] = h^2\sigma^2 \tag{60}$$

The iterations of the stochastic gradient descent method now reads as

$$\theta_{t+1} = \theta_t - \eta h(\theta_t - c_t) \tag{61}$$

where c_t is drawn i.i.d. from an unknown distribution with mean θ^* and variance σ^2. By inserting the update into Eq. 57, we obtain the expected loss after the SGD update:

$$\begin{aligned}
\mathrm{E}[\hat{f}(\theta_{t+1})|\theta_t] =& \mathrm{E}\left[\frac{1}{2}h((1-\eta h)\theta_t + \eta h c_t - c^\mu)^2\right] \\
=& \mathrm{E}\left[\frac{1}{2}h((1-\eta h)^2\theta_t^2 + 2(1-\eta h)\theta_t \eta h c_t \right. \\
& \left. -2(1-\eta h)\theta_t c^\mu + \eta^2 h^2 c_t^2 - 2\eta h c_t c^\mu + c^{\mu 2})\right] \\
=& \frac{1}{2}h\left[(1-\eta h)^2\theta_t^2 + \underbrace{2(1-\eta h)\theta_t \eta h \theta^* - 2(1-\eta h)\theta_t\theta^*}_{=-2(1-\eta h)^2\theta_t\theta^*}\right. \\
& \left. + \underbrace{\eta^2 + h^2(\theta^{*2}+\sigma^2) - 2\eta h(\theta^*)^2 + (\theta^{*2}+\sigma^2)}_{(1-\eta h)^2\theta^{*2}+\eta^2 h^2\sigma^2+\sigma^2}\right] \\
=& \frac{1}{2}h\left[(1-\eta h)^2(\theta_t-\theta^*)^2 + \eta 2h^2\sigma^2 + \sigma^2\right]
\end{aligned} \tag{62}$$

Note that c^μ and c_t correspond to different i.i.d. samples from the same distribution, and thus $\mathrm{E}(c^\mu c_t) = \theta^{*2}$.

4.1.2.2 *Optimal learning rate*

We can now derive the learning rate that optimizes the expected loss after an SGD update:

$$\eta_t^* = \arg\min_\eta \left[\frac{1}{2}h\left[(1-\eta h)^2(\theta_t-\theta^*)^2 + \eta 2h^2\sigma^2 + \sigma^2\right]\right] \tag{63}$$

We keep only terms in η and ignore all other constant terms:

$$\arg\min_\eta \left[\eta^2\left(h^2(\theta_t-\theta^*)^2 + h^2 + \sigma^2\right) - 2\eta\left(h^2(\theta_t-\theta^*)^2\right)\right] \tag{64}$$

It is a quadratic problem in η, and we obtain the solution by setting its derivative to zero:

$$0 = \frac{\partial}{\partial\eta}\left[\eta^2\left(h^2(\theta_t-\theta^*)^2 + h^2 + \sigma^2\right) - 2\eta\left(h^2(\theta_t-\theta^*)^2\right)\right] \tag{65}$$

$$\begin{aligned}
0 =& 2\eta\left(h^2(\theta_t-\theta^*)^2 + h^2 + \sigma^2\right) - 2\left(h^2(\theta_t-\theta^*)^2\right) \\
\eta_t^* =& \frac{1}{h}\frac{(\theta_t-\theta^*)^2}{(\theta_t-\theta^*)^2+\sigma^2}
\end{aligned} \tag{66}$$

Note that in the noiseless setting (that is, $\sigma^2 = 0$) the learning rate reduces to a newton step. The term $\frac{(\theta_t-\theta^*)^2}{(\theta_t-\theta^*)^2+\sigma^2}$ in the derived formula reduces the learning rate if the variance of the gradient is large i.e. when the sample gradients point in very different directions in parameter space. The reduction of the learning rate will be larger near

the optimum, since $(\theta_t - \theta^*)^2$ will then be small compared to the variance σ^2. On the other hand, we initially have relatively large learning rates since $(\theta_t - \theta^*)^2$ will be large. Hence, the derived learning rate is a decreasing learning rate schedule, but it anneals automatically. It is shown in appendix A of [30] that stochastic gradient with the derived learning rates almost surely converges to θ^*. Note, however, that the rate of convergence for stochastic gradient descent is at best $\mathcal{O}(\frac{1}{k})$ for strongly convex functions[6], and the derived learning rate schedule is still limited by this poor convergence rate. In contrast, one batch newton step would immediately jump to the minimum of the quadratic approximation, θ^*.

The derived learning rate in the one-dimensional case can be trivially generalized to d dimensions since we assume that the parameters are separable. We independently calculate a learning rate η^* for each dimension i. Alternatively, a more conservative approach is to derive a *global* learning rate:

$$\eta_t^* = \frac{\sum_{i=1}^{d} h_i^2(\theta_i^2 - \theta_i^*)^2}{\sum_{i=1}^{d} h_i^3(\theta_i^2 - \theta_i^*)^2 + h_i^3\sigma_i^2} \tag{67}$$

See appendix B of [30] for a complete derivation. The middle way between global and individual learning rates is to consider common learning rates for blocks of parameters. For instance, in GMLVQ we could have separate learning rates for prototype and transformation matrix variables[7].

4.1.2.3 *Approximations*

In practice, we do not know the values h, σ and $(\theta - \theta^*)^2$. However, based on Eq. 59 60, we can estimate these values from the gradient samples:

$$\eta_t^* = \frac{1}{h}\frac{(E[\nabla e^\mu])^2}{(E[\nabla e^\mu])^2 + \mathrm{Var}[\nabla e^\mu]} = \frac{1}{h}\frac{E[\nabla e^\mu])^2}{E[\nabla^2 e^\mu]} \tag{68}$$

For global learning rates, we assume that it is feasible to estimate the largest curvature h^+ of the block of parameters. We then obtain the bound:

$$\eta_t^* > \frac{1}{h^+}\frac{\sum_{i=1}^{d} h_i^2(\theta_i - \theta_i^*)^2}{\sum_{i=1}^{d} h_i^2(\theta_i - \theta_i^*)^2 + h_i^2\sigma_i^2} \tag{69}$$

$$= \frac{||E[\nabla e^\mu]||^2}{E\left[||\nabla e^\mu||^2\right]} \tag{70}$$

where the last step follows by the linearity of the expectation:

$$E[||\nabla e^\mu||^2] = E[\sum_{i=1}^{d}(\nabla e_i^\mu)^2] = \sum_{i=1}^{d} E[(\nabla L_i^\mu)^2] \tag{71}$$

6 The quadratic approximation is strongly convex

7 A natural choice since the proposed SGD method also uses separate learning rates for prototype and transformation matrix variables.

The adaptive learning rates thus depend on the inverse curvature and the squared expected gradient relative to the expected squared norm of the gradient. Recall that the latter term simply reduces the learning rate whenever is much variance in the gradient samples.

ADAPTIVE EXPONENTIAL MOVING AVERAGE We use an exponential moving average with time constant τ in order to estimate the expected gradient $g_i \approx E[\nabla e^\mu]$, the second moment of the gradient $v_i \approx E[(\nabla e^\mu)^2]$ and the squared length of the gradient vector $l \approx E[\|\nabla e^\mu\|^2]$:

$$g_i(t+1) = (1-\tau_i^{-1})g_i(t) + \tau^{-1}(\nabla e^\mu)_i \tag{72}$$

$$v_i(t+1) = (1-\tau_i^{-1})v_i(t) + \tau^{-1}(\nabla e^\mu)_i^2 \tag{73}$$

$$l(t+1) = (1-\tau_i^{-1})l(t) + \tau^{-1}\|\nabla e^\mu\|^2 \tag{74}$$

Note that v is only used in case of local element-wise learning rates, while l is only used for global learning rates. We adapt the time constant τ by the heuristic update rule:

$$\tau_i(t+1) = (1 - \frac{g_i^2(t)}{v_i(t)})\tau_i(t) + 1 \tag{75}$$

which slowly increases the memory size for a small parameter update, and which decays rapidly for large update steps i.e. whenever $\frac{g_i^2(t)}{v_i(t)}$ is close to one and a newton step is performed. Note that the heuristic update rule for the moving average eliminates the need to tune another hyper parameter (the memory size). Furthermore, notice that the update rule is scale invariant with respect to the curvature information.

CURVATURE ESTIMATION In the presented analysis we assumed that the problem is separable and thus that the Hessian matrix is diagonal. Although several methods exist for a diagonal Hessian approximation, we decide to use the diagonal approximation as proposed in [29] which was inspired by the method of Bordes et al. [4]. We start from a first-order Taylor approximation of the gradient around $\vec{\theta}_t$ and obtain the so-called secant equation:

$$\vec{\theta_{t+1}} - \vec{\theta}_t = H_{\vec{\theta}_t}^{-1}(\nabla f(\vec{\theta}_{t+1}) - \nabla f(\vec{\theta}_t)) \tag{76}$$

We replace the full Hessian inversion $H_{\vec{\theta}_t}^{-1}$ by a diagonal approximation $\hat{H}_t$:

$$\vec{\theta_{t+1}} - \vec{\theta}_t = \hat{H}_t(\nabla f(\vec{\theta}_{t+1}) - \nabla f(\vec{\theta}_t)) \tag{77}$$

which provides a unique diagonal Hessian approximation for each update step. However, in a SGD procedure we do not have access to the full gradients $\nabla f(\vec{\theta}_{t+1})$ and $\nabla f(\vec{\theta}_t)$, and we replace them by

the local gradients[8] $\nabla e^{\mu}(\vec{\theta}_{t+1})$ and $\nabla e^{\mu}(\vec{\theta}_t)$. In order to prevent sampling noise to enter the estimation process, we need to calculate the local gradients with respect to the very same example. The Hessian approximation thus doubles the number of gradient calculations.

In practice, we obtain the finite difference approximation of the Hessian by calculating two gradients at sample x^{μ} for parameters that differ by the typical gradient step $\delta = \bar{g}$:

$$\hat{h}_i^{\mu} = \left| \frac{\nabla e^{\mu}(\theta_i) - \nabla e^{\mu}(\theta_i + \delta_i)}{\delta_i} \right| \quad \text{with} \quad \delta_i = \bar{g}_i \tag{78}$$

We take the absolute value of the Hessian approximation to ensure that it is positive definite. As a consequence, we are able to deal with negative curvature; the direction of a step is simply reversed for parameters with negative second derivatives.

We use again our introduced adaptive moving average over the sample curvature:

$$\bar{h}_i = (1 + \tau^{-1})\bar{h}_i + \tau^{-1}\,\hat{h}_i \tag{79}$$

To further increase robustness, Schaul and LeCun suggest to take into account a moving average of the *variance* of the curvature v^{fd}, and then use a signal-to -noise ratio of the curvature estimates $\hat{h} = \frac{v^{fd}}{h}$. This strategy reduces the likelihood of underestimating the curvature and thus overestimating the learning rates. However, we found empirically that it is too conservative, in general. Therefore, we simply use the standard curvature estimation of Eq. 79.

In case of global or block learning rates, we simply take the largest curvature among all estimates: $h^{+} = \max_i h_i$. This strategy is quite conservative, therefore we also introduce a variant that takes the average block curvature $\bar{h}^{+} = \sum_{i=1}^{N} \frac{h_i}{N}$ as an estimate of the largest curvature. We refer to the first conservative variant as vSgd-b1, while the second variant is called vSgd-b2.

We also considered other strategies that might be effective to estimate the largest eigenvalues of the block parameters. For instance, in GMLVQ we know that the transformation matrix has a tendency to become low-rank at the optimum, and therefore the optimal sub-Hessian of the prototypes is also low-rank i.e. only few non-zero eigenvalues. A tight upper bound on the largest eigenvalue can thus be obtained by the trace[9] of the sub Hessian of the prototype i.e. only the diagonal terms needs to be calculated. It is, however, not clear if this approximation is also appropriate far from the optimum. Moreover, the sub-Hessian of the transformation matrix is not necessarily low-rank, and therefore the trace of the sub-Hessian would be a rather loose upper bound.

8 This trick was introduced in the online BFGS method of Schraudolph et al. [32]

9 which is equal to the sum of eigenvalues

Algorithm 2 Adaptive learning rates for SGD

Require: nrEpochs, n_0
  for $k = 1 \to n_0$ **do**
    Compute gradient $\nabla e^k(\vec{\theta})$
    $\bar{g} \to \bar{g} + \frac{\nabla e^k(\vec{\theta})}{n_0}$
    $\bar{v} \to \bar{v} + \frac{(\nabla e^k(\vec{\theta}))^2}{n_0}$
  end for
  $\bar{h} \to \left| \frac{\sum_{k=1}^{n_0} \nabla e^k(\theta) - \sum_{k=1}^{n_0} \nabla e^k(\theta+\bar{g})}{\bar{g}} \right|$

  for $j = 1 \to$ nrEpochs **do**
    for $\mu = 1 \to P$ **do**
      Compute gradients $\nabla e^\mu(\vec{\theta})$ and $\nabla e^\mu(\vec{\theta} + \bar{g})$
      for $i = 1 \to d$ **do**
        Compute curvature $h_i \to \left| \frac{\nabla e^\mu_{\theta_i} - \nabla e^\mu(\theta_i + \delta_i)}{\delta_i} \right|$
        Update moving averages
        $\bar{g}_i \to (1 - \tau_i^{-1})\bar{g}_i + \tau_i^{-1} \nabla e^\mu(\theta_i)$
        $\bar{v}_i \to (1 - \tau_i^{-1})\bar{v}_i + \tau_i^{-1} (\nabla e^\mu(\theta_i))^2$
        $\bar{h}_i \to (1 - \tau_i^{-1})\bar{h}_i + \tau_i^{-1} h_i$
        Determine learning rate $\eta_i = \frac{1}{\bar{h}_i} \frac{\bar{g}_i^2}{\bar{v}_i}$
        Update parameter $\theta_i = \theta_i + \eta_i \nabla e^\mu_{\theta_i}$
        Update memory size $\tau_i = (1 - \frac{\bar{g}_i^2}{\bar{v}_i})\tau_i + 1$
      end for
    end for
  end for

INITIALIZATION Before we start with the SGD optimization, we initialize the estimated quantities by computing the average over n_0 samples of the training set. The authors advise to incorporate a slow-start heuristic to keep the parameter small until the estimated quantities are stable. This can be realized by overestimating the variance v (or l in case of global learning rates) by a factor C. The authors suggest a heuristic rule of $C = \frac{d}{10}$ and show in the appendix D of [30] that the performance of the algorithm is not sensitive to the initialization of C.

4.1.3 *BFGS method*

To date, one of the most used quasi-Newton methods is the BFGS algorithm, that was discovered independently by Broyden [11], Fletcher [16], Goldfarb [17] and Shanno [35] in the 1970's. In this section we outline the BFGS algorithm, and apply the method to GMLVQ.

We start our derivation of the BFGS method from a quadratic model at the current iteration $\vec{\theta}_t$:

$$m_t(\vec{p}) = f(\vec{\theta}_t) + \nabla f(\vec{\theta}_t)\vec{p} + \frac{1}{2}\vec{p}^\top B_t \vec{p} \tag{80}$$

where B_t is a positive definite approximation of the Hessian that will be updated at every iteration. We use the minimizer of such a quadratic model:

$$\vec{p}_t = -B_t^{-1}\nabla f(\vec{\theta}_t) \tag{81}$$

as a search direction for finding the next iterate:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \eta_t \vec{p}_t \tag{82}$$

The step size η_t will be determined by a line search procedure that obeys the so-called Wolfe conditions for reasons that we will discuss later.

So far, the presented method is close to Newton's method since we have only replaced the true Hessian by an approximation B_t. The rationale behind quasi-Newton methods is to incrementally update the approximation rather than computing a new one at each iteration (as is the case for Newton's method). We will derive the update rule for the Hessian approximation B_{t+1} by imposing the condition that the new quadratic model

$$m_{t+1}(\vec{p}) = f(\vec{\theta}_{t+1}) + \nabla f(\vec{\theta}_{t+1})\vec{p} + \frac{1}{2}\vec{p}^\top B_{t+1}\vec{p} \tag{83}$$

matches the gradient of the cost function at $\vec{\theta}_t$ and $\vec{\theta}_{t+1}$. This latter condition is satisfied by construction, since $\nabla m_{t+1}(\vec{0}) = \nabla f(\vec{\theta}_{t+1})$. The first condition gives us:

$$\nabla m_{t+1}(-\eta_t \vec{p}_t) = \nabla f(\vec{\theta}_{t+1}) - \eta_t B_{t+1}\vec{p}_t \tag{84}$$

We rewrite this to the so-called *secant equation*:

$$B_{t+1}\vec{s}_t = \vec{y}_t \quad \text{or equivalently} \quad B_{t+1}^{-1}\vec{y}_t = \vec{s}_t \tag{85}$$

where $\vec{s}_t = \vec{\theta}_{t+1} - \vec{\theta}_t$ and $\vec{y}_t = \nabla f(\vec{\theta}_{t+1}) - \nabla f(\vec{\theta}_t)$ denote the recent change in parameter and gradient space, respectively. The two versions of the secant equation give rise to two different quasi-Newton methods, the DFG and BFGS update. The former was already proposed in the mid 1950's and essentially keeps an approximation of the Hessian B_{t+1}. On the other hand, the BFGS method directly updates an approximation of the *inverse* of the Hessian matrix $C_{t+1} = B_{t+1}^{-1}$, and will be the focus

In order to ensure that C_{t+1} remains positive definite[10], the displacement $\vec{s}_t$ and change in gradient $\vec{y}_t$ must satisfy the *curvature condition*:

$$\vec{s}_t\,\vec{y}_t > 0 \tag{86}$$

10 As should be easily seen by multiplying the second part of Eq. 85 from the left by $\vec{y}$. Note that if we ensure the approximation of the inverse of the Hessian C_{t+1} to be positive definite, then the Hessian approximation B_{t+1} itself is also positive definite.

For convex cost functions the curvature condition always hold, while for non-convex cost function we need to impose Wolfe conditions on the line search procedure. We refer the interested reader to [26] for more details.

In the following we assume that the curvature conditions hold, and at least one solution to the secant equation exist. In fact, it turns out that there are infinitely many B_{t+1} that satisfy the secant equation. To uniquely determine B_{t+1} we impose the additional constraint that the B_{t+1} is closest to B_t with respect to a weighted Frobenius norm. In other words, we solve the following minimization problem:

$$\begin{aligned} \underset{C_{t+1}}{\text{minimize}} \quad & \|C_{t+1} - C_t\|_W \\ \text{subject to} \quad & C = C^\top \cdot \\ & C\vec{y}_t = \vec{s}_t \end{aligned} \tag{87}$$

where the weighted Frobenius norm is defined as:

$$\|A\|_W = \|W^{\frac{1}{2}} A W^{\frac{1}{2}}\|_F \tag{88}$$

The matrix W is chosen to be the inverse G_t^{-1} of the average Hessian:

$$G_t = \int_0^1 \nabla^2 f(\vec{\theta}_t + \tau\eta_t\vec{p}_t)\,d\tau \tag{89}$$

in order to make the optimization problem scale-invariant. The BFGS update rule is the closed form solution to this minimization problem:

$$C_{t+1} = (I - \rho_t \vec{s}_t \vec{y}_t^\top) C_t (I - \rho_t \vec{s}_t \vec{y}_t^\top) + \rho_t \vec{s}_t \vec{s}_t^\top \tag{90}$$

We summarize the BFGS method in Algorithm 3. Note that we initialize the approximation C_0 by the identity matrix I. The BFGS method has several advantages over the well-known Newton's method. First of all, the iterations can be performed at a cost of $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ for Newton's method (due to matrix inversion). Furthermore, it does not require to calculate the second-derivatives explicitly, which i) can be hard to analytically derive[11] and ii) is computationally expensive to evaluate. Moreover, the BFGS approximation of the Hessian is positive definite by construction, and therefore we do not have to deal with negative curvature. Finally, the rate of convergence for BFGS is still superlinear and is only slightly worse than the quadratic convergence rate of Newton's method.

4.1.4 *l-BFGS*

Limited memory BFGS (l-BFGS) [26] is useful for large-scale optimization problems whose Hessian is too large to store and manipulate. So,

11 For example, see the second derivatives of GMLVQ in appendix A.2

Algorithm 3 BFGS method

Require: Line search $linemin$ obeying Wolfe conditions
Require: Stopping criterion ϵ
$C_0 = I$
while $\|\vec{\theta}_t - \vec{\theta}_{t-1}\| > \epsilon$ **do**
$\vec{p}_t = -C_t \nabla f(\vec{\theta}_t)$
$\eta_t = linemin(f, \vec{\theta}_t, \vec{p}_t)$
$\vec{s}_t = \eta_t \vec{p}_t$
$\vec{\theta}_{t+1} = \vec{\theta}_t + \vec{s}_t$
$\vec{y}_t = \nabla f(\vec{\theta}_t) - \nabla f(\vec{\theta}_{t-1})$
$\rho_t = \frac{1}{\vec{s}_t^\top \vec{y}_t}$
$C_{t+1} = (I - \rho_t \vec{s}_t \vec{y}_t^\top) C_t (I - \rho_t \vec{s}_t \vec{y}_t^\top) + \rho_t \vec{s}_t \vec{s}_t^\top$
$t = t + 1$
end while

instead of storing a full $N \times N$ matrix (as is the case for BFGS), we only store a few N-dimensional vectors that implicitly represent the Hessian approximation. We start our derivation of the l-BFGS method by recalling the BFGS update:

$$C_{t+1} = V_t^\top C_t V_t + \rho_t \vec{s}_t \vec{s}_t^\top \quad \text{where} \quad \rho_t = \frac{1}{\vec{s}_t^\top \vec{y}_t} \quad \text{and} \quad V = I - \rho_t \vec{s}_t \vec{y}_t^\top \tag{91}$$

By repeated application of this update rule we find that the inverse Hessian approximation can be written as:

$$\begin{aligned} C_t =& (V_{t-1}^\top \dots V_{t-m}^\top) C_0 (V_{t-1} \dots V_{t-m}) \\ &+ \rho_{t-m} (V_{t-1}^\top \dots V_{t-m+1}^\top) \vec{s}_{t-m} \vec{s}_{t-m}^\top (V_{t-1} \dots V_{t-m+1}) \\ &+ \rho_{t-m} (V_{t-1}^\top \dots V_{t-m+2}^\top) \vec{s}_{t-m} \vec{s}_{t-m}^\top (V_{t-1} \dots V_{t-m+2}) \\ &\vdots \\ &+ \rho_{t-m} \vec{s}_{t-1} \vec{s}_{t-1}^\top \end{aligned} \tag{92}$$

where C_0 is the initial approximation of the inverse Hessian. This trick enables us to reimplement the standard BFGS update by only storing all $\{\vec{s}_t, \vec{y}_t\}$ pairs[12]. However, this approach is more expensive in terms of memory usage than BFGS if the number of iterations exceeds $\frac{N}{2}$[13]. Therefore, the strategy of l-BFGS is to approximate the inverse Hessian by only using the $\{\vec{s}_t, \vec{y}_t\}$ information of the past m iterations.

There is no need to explicitly reconstruct a Hessian approximation C_t because we only need the search direction $C_t \nabla f(\vec{\theta}_t)$. An efficient procedure, which only uses inner products and vector additions, can

12 We also have to choose the same initial C_0, then
13 Due to symmetry of the Hessian we only need to store half of the matrix

Algorithm 4 l-BFGS search direction computation

Ensure: $\vec{r} = C_t \nabla f(\vec{\theta}_t)$
  $\vec{q} = \nabla f(\vec{\theta}_t)$
  for $i = t-1 \rightarrow t-m$ **do**
    $\alpha_i = \rho_i \vec{s}_i^\top \vec{q}$
    $\vec{q} = \vec{q} - \alpha_i \vec{y}_i$
  end for
  $\vec{r} \rightarrow C_0 \vec{q}$
  for $i = t-m \rightarrow t-1$ **do**
    $\beta = \rho_i \vec{y}_i \vec{r}$
    $\vec{r} = \vec{r} + \vec{s}_i(\alpha_i - \beta)$
  end for

be derived to compute this search direction. The algorithm is outlined in Algorithm 4. Note that the procedure only requires mN multiplications, if we ignore the $C_0 \vec{q}$ multiplication. For diagonal C_0 this would only require N additional multiplications. Another advantage of the derived procedure is that the initial matrix approximation C_0 can be chosen differently for each iteration. We use $C_0 = \frac{\vec{s}^t o p_{t-1} \vec{y}_{t-1}}{\vec{y}_{t-1}^\top \vec{y}_{t-1}} I$ since it argued to be effective, in practice.

We summarize the l-BFGS method in Algorithm 5. In practice, the method works surprisingly well for even modest values of m, and it often suggested to set $m \in [5, 30]$. It can further be argued that the l-BFGS has the advantage over BFGS that it only uses *relatively new* curvature information. The inverse Hessian approximation of BFGS contains information from *all* previous iterations, which might be problematic if curvature of the cost function differs significantly over the optimization trajectory. Nevertheless, the l-BFGS does not obtain the super-linear convergence of BFGS, but still achieves an acceptable linear convergence [26]. We emphasize that the main advantage is that l-BFGS scales linearly in the number of parameters which makes it suitable for large scale problems.

4.2 EXPERIMENTS

In the following we evaluate the performance of the presented optimization methods on artificially generated datasets. Section 4.2.1 provides necessary background information on the generated dataset, parameter initialization and optimization settings. Section 4.2.2 investigates the effect of increasing the dimensionality on the performance of the optimization methods, while section 4.2.3 addresses the effect of increasing the size of the dataset.

Algorithm 5 l-BFGS method

Require: Memory level m
Require: Line search $linemin$ obeying Wolfe conditions
Require: Stopping criterion ϵ
  while $||\vec{\theta}_t - \vec{\theta}_{t-1}|| > \epsilon$ **do**
    Set $C_0 = \frac{\vec{s}^t o p_{t-1} \vec{y}_{t-1}}{\vec{y}_{t-1}^\top \vec{y}_{t-1}} I$
    Determine $\vec{p}_t$ by Algorithm 4
    $\eta_t = linemin(f, \vec{\theta}_t, \vec{p}_t)$
    $\vec{s}_t = \eta_t \vec{p}_t$
    $\vec{\theta}_{t+1} = \vec{\theta}_t + \vec{s}_t$
    $\vec{y}_t = \nabla f(\vec{\theta}_t) - \nabla f(\vec{\theta}_{t-1})$
    if $t > m$ **then**
      Discard $\{\vec{s}_{t-m}, \vec{y}_{t-m}\}$ pair from memory
    end if
    Save $\{\vec{s}_t, \vec{y}_t\}$ pair
    $t = t + 1$
  end while

4.2.1 *Preliminaries*

For all performed experiments we have made the following decisions with respect to the datasets, parameter initialization and optimization settings.

4.2.1.1 *Datasets*

We construct a binary classification problem by drawing samples from a multivariate Gaussian distribution $\mathcal{N}_k(c, \Sigma)$ with probability density function:

$$p(\vec{x}) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp((\vec{x} - \vec{c})^\top \Sigma^{-1} (\vec{x} - \vec{c})) \tag{93}$$

We center the normal distributions at $c_1 = [1, 1, \ldots, 1]$ and $c_2 = [-1, -1, \ldots, -1]$ for the first and second class, respectively. For both classes we use the *same* randomly generated covariance matrix Σ. In order to ensure positive definiteness of the covariance matrix, we randomly generate a square matrix L with elements $L_{ij} \in \mathcal{U}(-1, 1)$, and subsequently define $\Sigma = L^\top L$. Note that we do not normalize the covariance matrix, and thus different instantiations will lead to covariance matrices of different norm[14]. Consequently, the overlap between the two classes varies over the constructed data sets.

14 However, it will have a maximum Frobenius norm of N^2

4.2.1.2 *Initialization of parameters*

We evaluate the performance of the optimization methods on the generated datasets for N_{init} initializations of the parameters. We decide to use one prototype per class and initialize them as the class-conditional means. However, in order to construct different prototype initializations, we add a small perturbation $\vec{v} \in \mathcal{U}(-1,1)^N$ to the prototypes. Furthermore, we randomly initialize the rectangular transformation matrix $\Omega \in \mathcal{R}^{M \times N}$ by drawing each element Ω_{ij} from a uniform distribution $\mathcal{U}(-1,1)$. In our experiments we construct datasets of dimension 10, 50 and 100, and thereby use rectangular transformation matrices of 5×10, 30×50 and 60×100, respectively.

4.2.1.3 *Optimization settings*

In the following we give an overview of the used settings in the presented optimization methods.

SGD We use a grid search to determine learning rates for the SGD procedure. We consider all combinations of the following learning rates:

$$\eta_W \in \{100, 10, 5, 1, 0.1, 0.05, 0.01, 0.005, 0.001, 5e-4, 1e-4\} \quad (94)$$

$$\eta_\Omega \in \{100, 10, 5, 1, 0.1, 0.05, 0.01, 0.005, 0.001, 5e-4, 1e-4\} \quad (95)$$

and pick the learning rates η_{max} with the lowest cost function value after *two epochs*. We then anneal the learning rate using the following schedule:

$$\eta(t) = \frac{\eta_{max}}{1 + \frac{t}{\text{epochs}}} \quad (96)$$

Note that learning rate will eventually be half of the initial learning rate. We decide to run SGD for 100 epochs.

ADAPTIVE LEARNING RATES FOR SGD We initialize the statistics of the adaptive learning rates over $n_0 = 100$ samples. Moreover, we use a slow start heuristic of $C = 2$. We run the following three variants of the adaptive learning rates for 100 epochs:

VSGD-L Use the local gradient variance and local hessian estimate, resulting in the learning rate $\eta_i^* = \frac{(\bar{g}_i)^2}{\bar{h}_i \bar{v}_i}$.

VSGD-B1 We distinguish two blocks of parameters: prototype and transformation matrix variables. We then use the block-specific gradient variance and the maximum curvature of the block of parameters, leading to $\eta^* = \frac{\sum_i (\bar{g}_i)^2}{h^+ \bar{l}}$ with $h^+ = \max_i \bar{h}_i$.

VSGD-B2 We also use block-specific gradient variance, but estimate block curvature by an average $\bar{h}^+ = \frac{1}{N}\sum_{i=1}^{N} \bar{h}_i$ over the diagonal Hessian elements, leading to the block-specific learning rate $\eta^* = \frac{\sum_i (\bar{g}_i)^2}{\bar{h}^+ \bar{l}}$.

BFGS The BFGS method is a batch optimization algorithm which makes it difficult to directly compare the parameter updates with the stochastic optimization methods. We think, however, that one iteration of a batch method is somewhat similar to one epoch since we use gradient information with respect to the same number of examples. Therefore, we limit the maximum number of iterations of the BFGS method to 100 iterations. We stop, however, the optimization procedure whenever the parameter update becomes smaller than $\epsilon = 1e-6$.

L-BFGS We use the same number of iterations and the same stopping criteria as BFGS. In addition, we use a memory level of $m = 20$.

4.2.2 *Effect of dimensionality* N

We perform the experiments for 10 datasets[15] of dimension 10, 50 and 100. For each generated dataset we consider 5 different initializations of the parameters. In total, we thus have 50 runs for an optimization method.

4.2.2.1 *Training error*

In order to obtain the general performance of the optimization method, we simply take the average over 50 runs. Note that each generated dataset has a different minimum cost function value, and therefore it is not very useful to take into account the variance. Furthermore, we restrict our analysis to the cost function values, and do not really care about the classification error[16]. In the following we refer to training error

Fig. 9 compares the development of the average cost function value for the described optimization methods. From the graph we can see that the performance of vSGD-b1 is extremely poor. Its strategy to take the maximum of a block of curvature estimates is too conservative. Note that this estimate becomes more and more conservative when we increase the dimension N. With more parameters per block there is probably a bigger spread of the curvature estimates i.e. the condition number increases. This means that the largest curvature will be more and more an overestimation for the curvature of the

15 For $N = 100$ we use 8 datasets due to some memory issues of my laptop

16 It is outside the scope of this thesis to investigate the discrepancy between cost function and classification error.

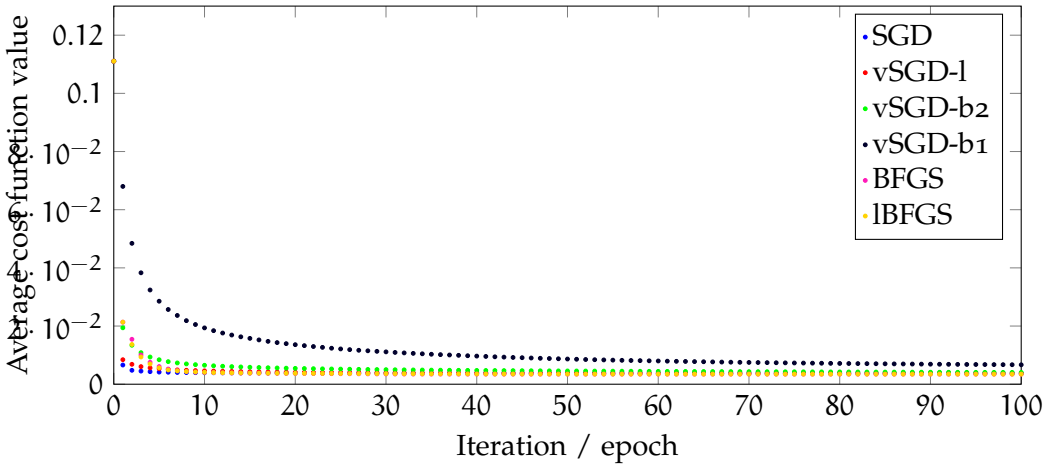
N	SGD	vSGD-l	vSGD-b1	vSGD-b2	BFGS	l-BFGS
10	7.3374	7.5573	8.1204	13.3551	6.9239	6.9242
50	7.1810	7.6663	9.8354	128.7715	6.8582	6.8480
100	7.5735	8.2114	16.8653	325.9584	7.1370	7.0587

Table 1: Average cost function value ($\cdot 10^{-3}$) on *training* data

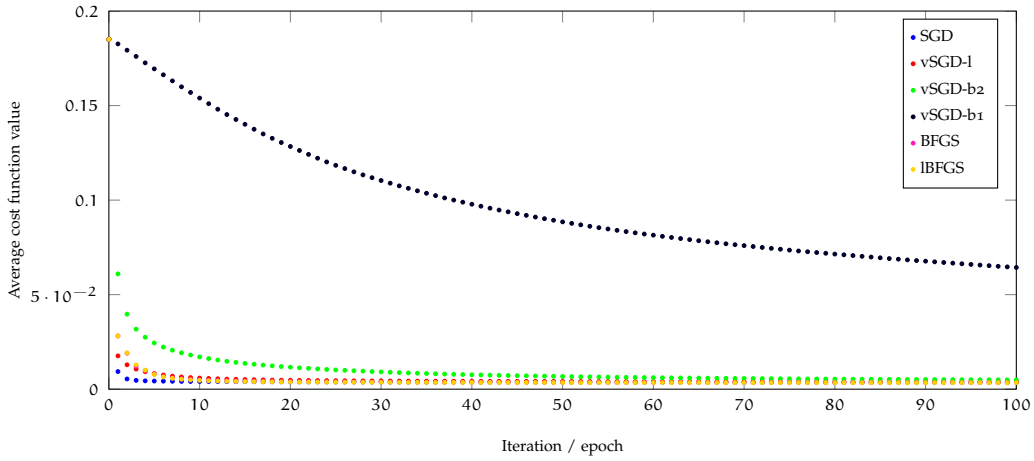
other parameters. Indeed, Fig. 9 c) clearly illustrates that increasing the dimension N deteriorates the performance. Taking the average of the curvature estimates, as realized in vSGD-b2, results in better performance. However, increasing the dimension deteriorates the performance in comparison with the other optimization methods. One particular reason might be that the condition number of the block of parameter increases, and therefore a block-specific learning rate is not sufficient to obtain good performance. The vSGD-l is the only variant of the adaptive learning rate variant that competes with the other methods.

From Fig. 9 it is hard to compare the SGD, vSGD-l, BFGS and lBFGS method, and at first glance the performance seems similar. We further inspect the results by presenting the average cost function value at the end of training procedure in Table 1. The batch methods, BFGS and l-BFGS, clearly outperform the stochastic optimization methods in terms of cost function value on the *training data*. Interestingly, the limited memory version of BFGS is slightly better than BFGS for datasets of dimension 50 and 100. Furthermore, the results for the stochastic optimization methods indicate that the greedy grid search to determine the learning rates slightly outperforms the adaptive local learning rate of the vSGD-l. In this light, it is interesting that the difference in cost function value is somewhat similar for all N-dimensional datasets. We would expect that for high-dimensional datasets it is beneficial to use the individual adaptive learning rate of vSGD-l since it can reduce ill-conditioning within a block of parameters. However, it only takes into account a diagonal Hessian approximation, while there might be strong influence of the off-diagonal entries.

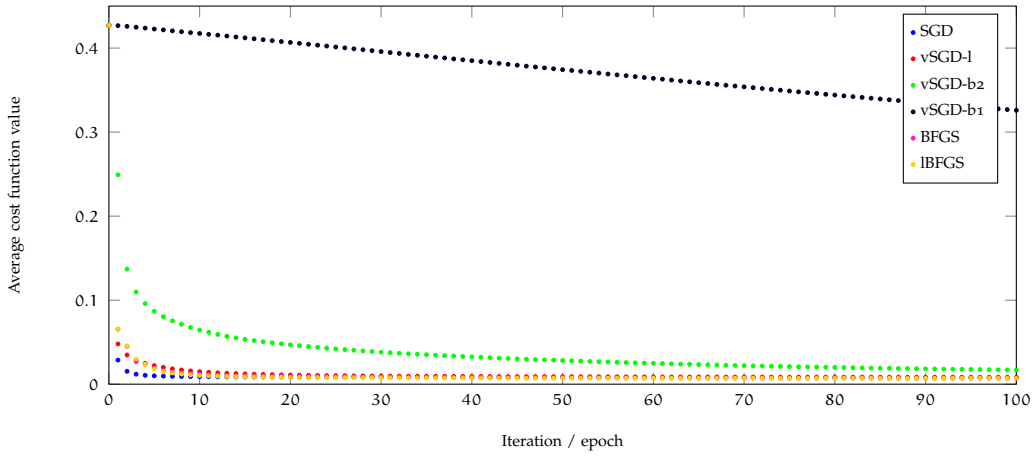
Fig. 10 provides histograms of the chosen learning rates for the SGD method. From the graphs we see that there is much variation in the learning rates for the prototypes. The most chosen learning rate is 1, but the learning rate ranges from $5e-4$ to 10. There is less variation in the learning rates for the transformation matrix. Only the very large learning rates 1-100 are used. Especially for dimension $N = 100$ we see that only the largest learning rate $\eta_\Omega = 100$ is chosen. This might suggest that the performance of the SGD method could be further improved by including even larger learning rates in the grid search. However, Fig. 9 c) illustrates that the learning rates are large enough since the learning curve shows a fast initial convergence. In



(a) N = 10

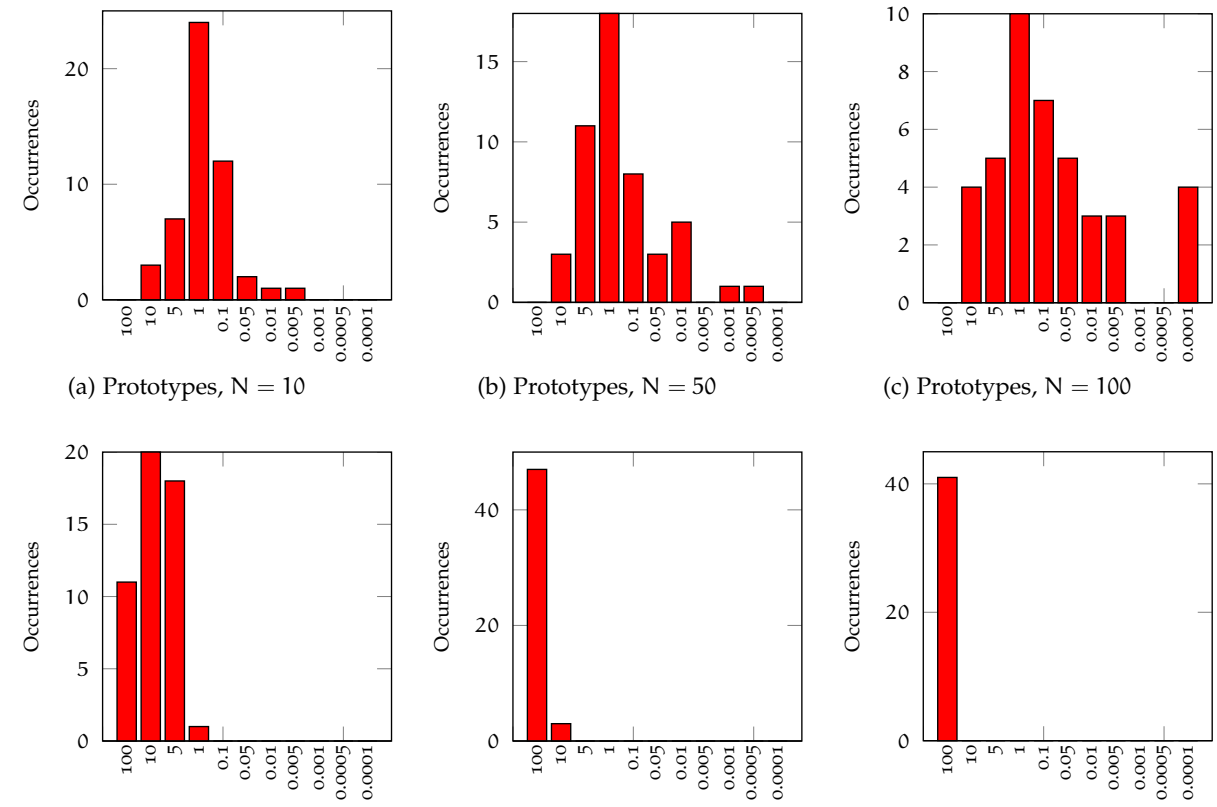


(b) N = 50



(c) N = 100

Figure 9: Development of the average cost function value on *training* data of dimension N.



(a) Prototypes, N = 10 (b) Prototypes, N = 50 (c) Prototypes, N = 100

(d) Transformation matrix, N = 10 (e) Transformation matrix, N = 50 (f) Transformation matrix, N = 100

Figure 10: Histogram of the chosen learning rates for the SGD method.

general, the results indicate that is hard to predict learning rates beforehand, and a grid search is thus really necessary to obtain good performance. Our findings are in conflict with the conventional wisdom for choosing good learning for GMLVQ; Schneider et al. [31] uses small learning rates in the order of $1e-3$ and suggest to set the learning rate of the transformation matrix an order of magnitude smaller than the learning rate of the prototypes.

4.2.2.2 *Generalization error*

As mentioned before, in machine learning it is not the performance on training data but on *test* data that matters. Therefore, we draw 200 data points from the same distribution, and compare the performance of the optimization methods on this unseen dataset. Fig. 12 shows the the average cost function value on the test set over training time. Note that we only illustrate the cost function value in the interval $[0, 10]$ to be able to compare the best optimization methods. In general, the performance on test data seems to be fairly similar to the performance on training data. That is to say, we do not experience over fitting behaviour in our experiments.

N	SGD	vSGD-l	vSGD-b1	vSGD-b2	BFGS	l-BFGS
10	7.6416	7.9065	8.6058	14.8184	6.9454	6.9457
50	7.3710	8.2950	12.6593	143.9980	6.9206	6.9063
100	9.1212	11.7436	30.1836	337.8680	7.7397	7.4546

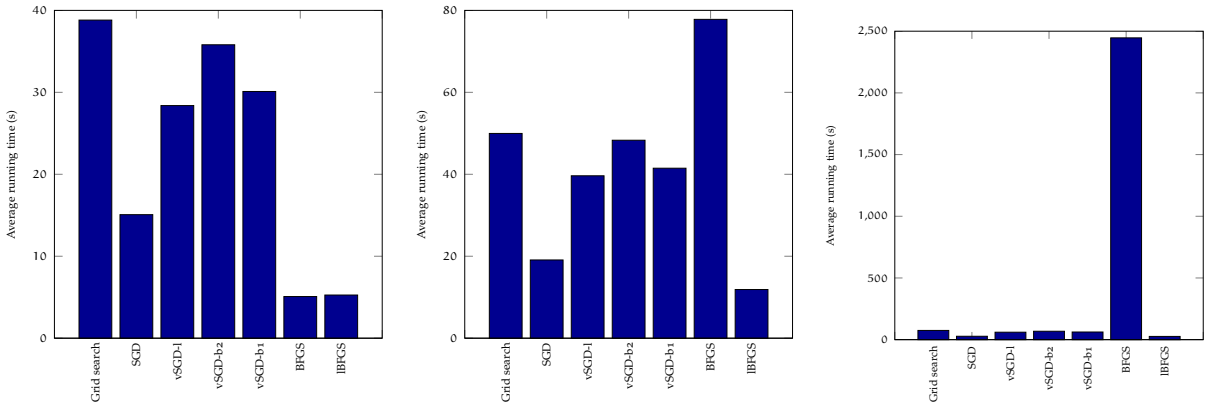
Table 2: Average cost function value ($\cdot 10^{-3}$) on *test* data

The SGD method with greedy search for the learning rates is superior in the initial phase (up to 10 epochs). As a stochastic optimization method it exploits redundancies in the data sets by performing many cheap, but noisy, gradient updates. In this light, it is somewhat surprising that the vSGD-l method shows slow initial convergence. It could be either due to the slow-start heuristic or the adaptive learning rates that are too conservative. Note that the noise in the gradient is also the reason for slow convergence of the stochastic methods in the final phase. In contrast, it is precisely this final phase where the second-order batch methods flourish (at least in optimizing the training error). From the plots in Fig. 12 we see that the l-BFGS outperforms the SGD method also in terms of generalization performance, after 20 iterations. Hence, the BFGS methods are not only better in optimizing the training error, but it actually results in better generalization errors.

To further analyse the results, we present the average generalization error at the end of training in Table 2. From a comparison with training error in Table 1, it follows (correctly) that the average cost function value on test data is slightly higher than for training data, in general. The most striking observation that emerge from this comparison is that for $N = 100$ the generalization error of the stochastic methods is *much worse* than its training error. For SGD and vSGD-l the difference is respectively 1.5477 and 3.5322, while it is only 0.3959 for l-BFGS. It is in conflict with the wide belief that stochastic optimization methods often yield much better generalization performance than batch optimization methods.

4.2.2.3 *Running time*

It is not only important for the optimization methods to reach low generalization error, they should also run within reasonable time. We present the average running time of the described methods in Fig. 11. From the histograms we see that the execution time of the adaptive learning rates methods is twice as long as the execution time for plain SGD. It can be ascribed to the local curvature estimation that needs an extra gradient evaluation at each stochastic update. However, the SGD method requires a grid search that is computationally more expensive than the stochastic methods. In total, the SGD method is thus slightly more expensive than the adaptive learning rates for SGD. Fur-



(a) N = 10 (b) N = 50 (c) N = 100

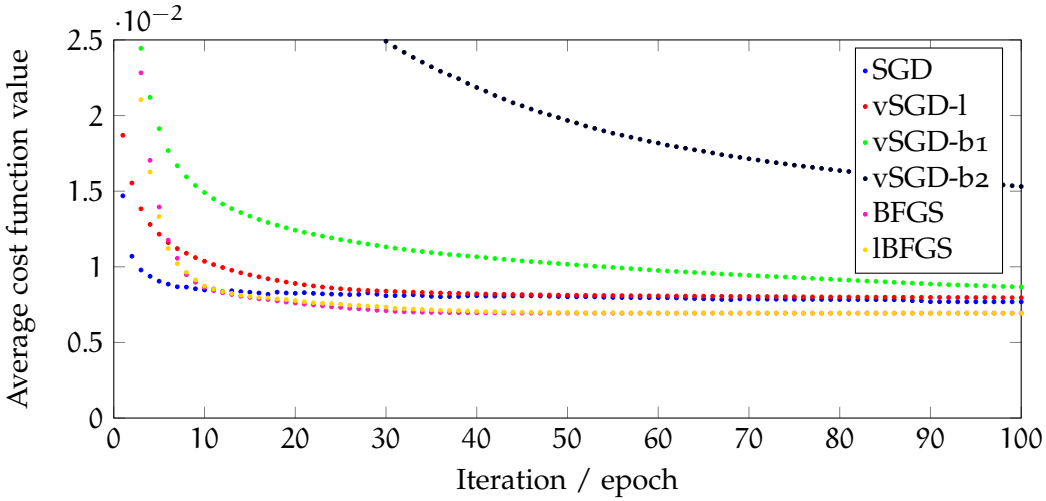
Figure 11: Histograms of the average runtime for datasets of dimension N

thermore, note that increasing the dimensionality does not have any influence on the relative running time of the stochastic methods. The adaptive learning rates for SGD treats each dimension independently, and thus scales linearly in computational cost.

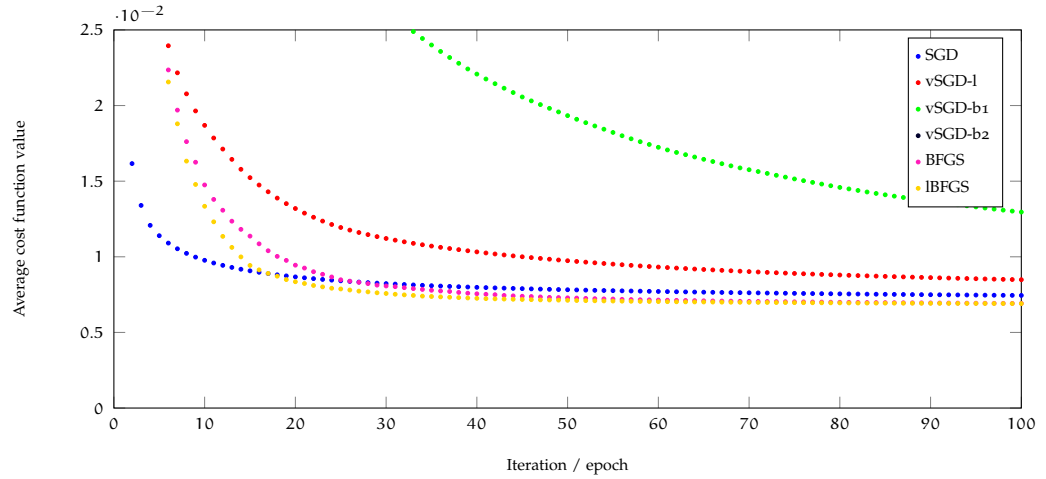
For 10-dimensional datasets, the computation time of BFGS and l-BFGS is similar. As expected, increasing the dimensionality quickly leads to a dramatic increase of the computation time of BFGS. For $N = 100$, the average running time is about 40 minutes while it is less than 2 minutes for the other optimization methods. The reason is that for a 100 dimensional dataset with a 100×60 transformation matrix we already have 6200 parameters, and therefore BFGS has to maintain a 6200×6200 approximation of the inverse of the Hessian. The limited memory version does not suffer from quadratic scaling in the number of dimension, and therefore has a much lower running time. The most important observation from the histograms is that the running time of l-BFGS is about four times less than the running time of all other optimization methods. For small datasets, the l-BFGS clearly outperforms the other optimization methods in all aspects.

4.2.2.4 *Rank of relevance matrix*

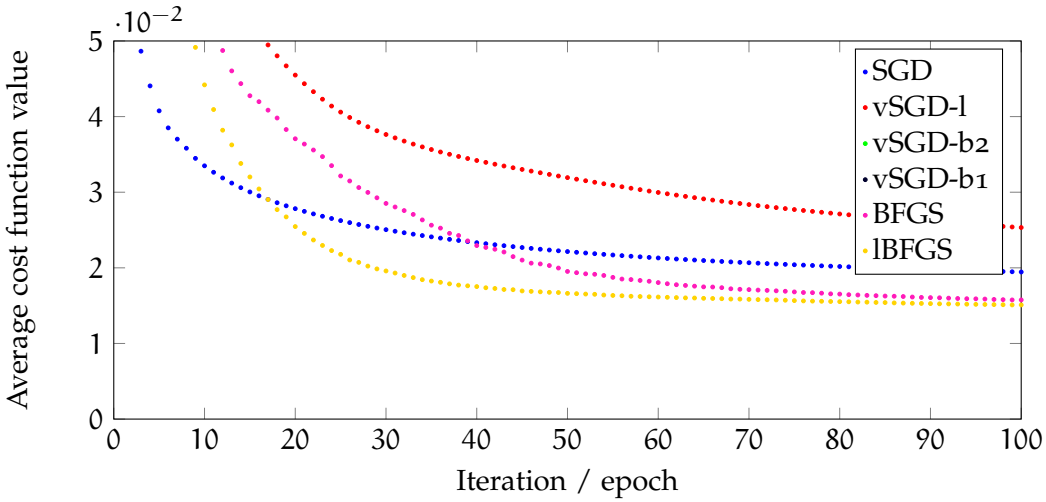
In section 3.1.1 we argued that the relevance matrix has a tendency to become low-rank. In order to verify our theoretical arguments, we study the eigenvalue profile of the relevance matrix. We decide to use the results from the l-BFGS method since it showed the best performance, in general. Nevertheless, the other optimization methods should give similar results. Fig. 13 shows the average eigenvalue profile of the optimal relevance matrix Λ for N-dimensional datasets. From the histograms we observe that, indeed, the relevance matrix has only one dominant eigenvalue. Note that a $M \times N$ rectangular Ω limits the rank of the Λ matrix to at most M. Hence, the first 5, 20



(a) N = 10

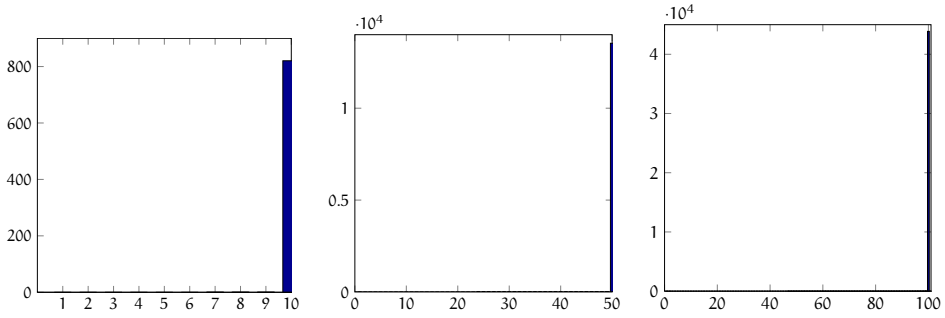


(b) N = 50



(c) N = 100

Figure 12: Development of the average cost function value on test data of dimension 10, 50 and 100.



(a) N = 10 (b) N = 50 (c) N = 100

Figure 13: Average eigenvalue profile of the Λ matrix for datasets of dimension N

and 40 eigenvalues are zero in the histograms of $N = 10$, $N = 50$ and $N = 100$, respectively. However, we clearly observe that the other eigenvalues are also very small in comparison with the largest eigenvalue, and thus the relevance matrix has rank one for all N-dimensional datasets.

The experiments confirm that GMLVQ only measures the distances along one direction, even for very high dimensional problems. In our artificially generated datasets this direction should equal the eigenvector of the covariance matrix with the *smallest* eigenvalue, since it is this direction that separates the classes with largest margin. Although other directions are also able to separate the two classes, GMLVQ eliminate these direction because it would deteriorate the cost function value. This also implies that the optimal cost function value is more or less similar for all N-dimensional problems. Indeed, Table 1 shows that cost function value is of the same order of magnitude. On the other hand, the *initial* cost function value deteriorates whenever we increase the dimension N, as can be observed from Fig. 9. The extra degrees of freedom in the transformation matrix causes a random initialization to be further from optimal, on average.

4.2.3 *Effect of number data points* P

In this section we investigate the effect of increasing the dataset size on the performance of the optimization methods. We perform the experiments for datasets of size 50, 100, 200, 500, 1000, 2000. For each dataset size P, we generate 10 datasets, and run the optimization methods for 5 parameter initializations. We fix the dimension $N = 25$, and use transformation matrices of 10×25. Furthermore, we exclude the νSGD $-$ b1 and νSGD $-$ b2 method in the evaluation due to poor performance as shown in the previous section.

P	SGD	vSGD-l	BFGS	l-BFGS
50	7.2607	7.4954	6.8852	6.8597
100	7.9393	8.5018	7.2895	7.2791
200	7.0616	7.2344	6.9658	6.9644
500	6.8499	7.1870	6.7855	6.8031
1000	6.8656	7.1446	6.8074	6.8583
2000	7.3259	7.9645	7.1304	7.3483

(a) **Training**

P	SGD	vSGD-l	BFGS	l-BFGS
50	8.3427	8.9528	7.1703	7.0773
100	10.2923	11.4266	7.7667	7.7207
200	7.1730	7.3801	7.0461	7.0464
500	6.8622	7.3117	6.7888	6.8086
1000	6.8946	7.1570	6.8356	6.8918
2000	7.2573	7.7991	7.0910	7.2748

(b) **Test**

Table 3: Average cost function value ($\cdot 10^{-3}$) at the end of training for a) training set and b) test set. The training set is of increasing size P.

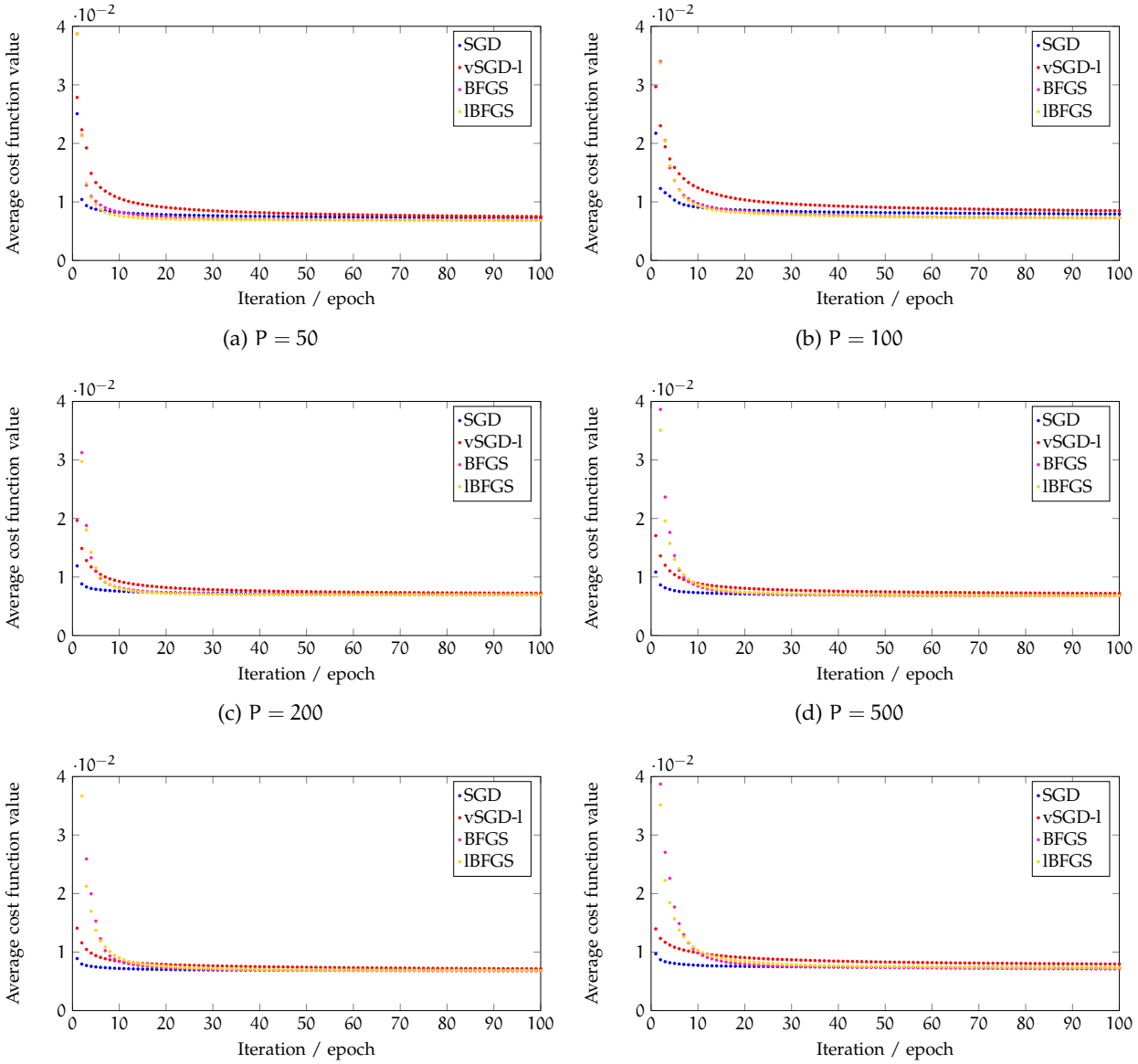
4.2.3.1 *Training error*

We first present the results of the optimization methods in terms of performance on training data. Fig. 14 shows the average optimization curve for datasets of size P. From these plots it is hard to compare the performance in the final convergence phase. Therefore, we also present the average cost function value at the end of training in Table 3 a). For small datasets, containing only 50 or 100 examples, we observe that BFGS and l-BFGS outperforms plain SGD. However, this difference becomes smaller when the dataset size increases. There might be two reasons at play, here. First of all, the updates of SGD are independent of the size of the dataset, and thus relatively more parameter updates are performed when the dataset increases. Note that this effect is clearly visible in Fig. 14. For increasing datasets the batch methods needs more and more iterations to obtain similar performance; More than 20 iterations are needed when $P = 2000$, while clearly less than 10 are sufficient for $P = 50$. Second, the grid search for learning rates is determined over two epochs. For larger datasets these learning rates are thus determined with respect to more parameter updates. This will lead to better (less greedy) learning rates, in general.

Note that the performance of vSGD-l does not get better for increasing datasets. We speculate that the adaptive learning rates are too conservative. Furthermore, what is interesting in Table 3 a) is that l-BFGS slightly outperforms BFGS for small datasets i.e. $P = 50$ and $P = 100$. On the other hand, for large datasets BFGS clearly outperforms l-BFGS.

4.2.3.2 *Generalization error*

To determine the generalization performance, we evaluate the cost function on 200 samples from the same distribution. We present the



(a) P = 50

(b) P = 100

(c) P = 200

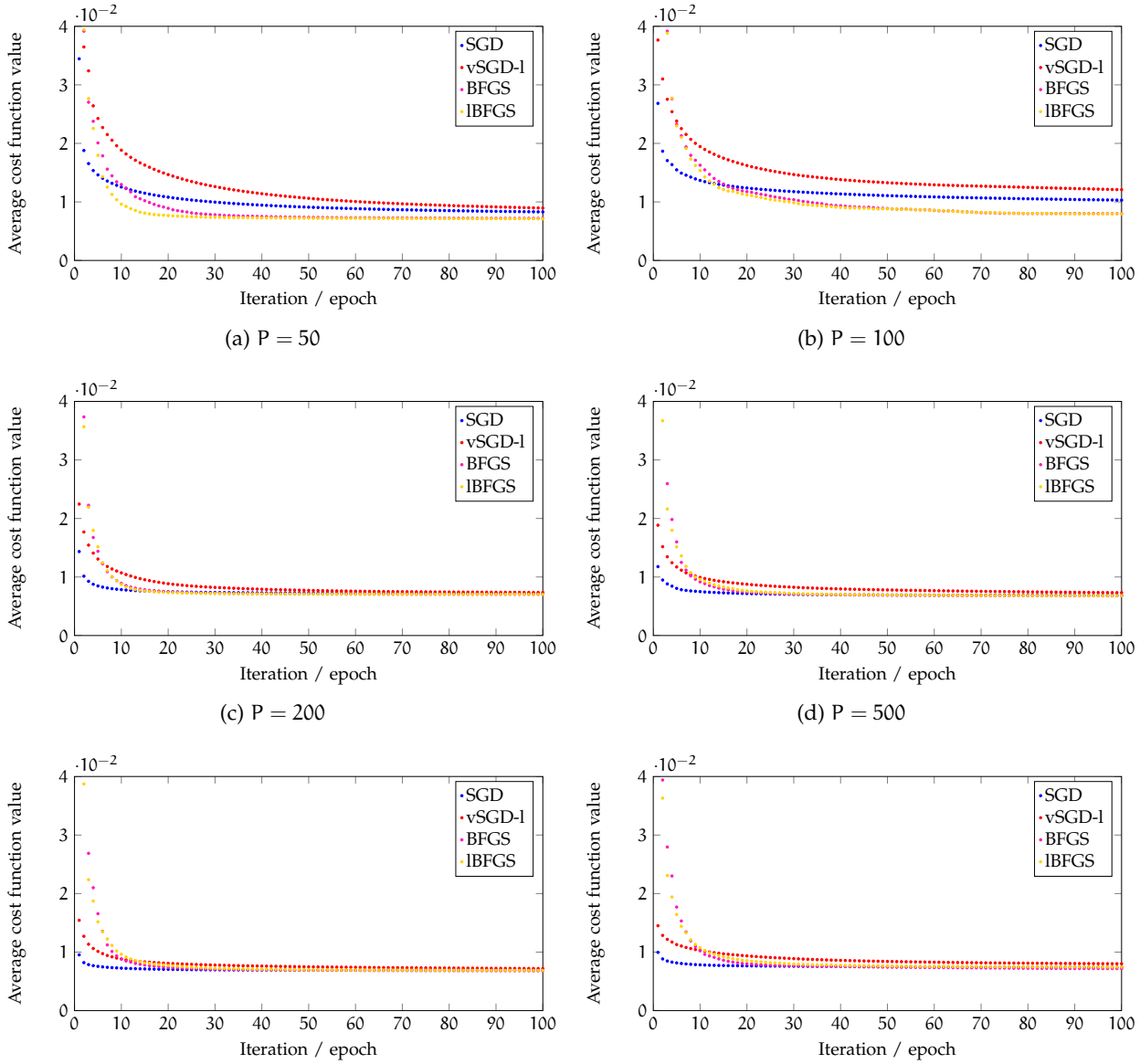
(d) P = 500

(e) P = 1000

(f) P = 2000

Figure 14: Average optimization curve for datasets of size P.

average learning curves in Fig. 15, and provide the average generalization error at the end of training in Table 3 b). The main observation is that better training error also results in better generalization error. Interestingly, for small datasets (P = 50 and P = 100) the generalization performance of the stochastic methods is much worse than (l-)BFGS. For example, we see that for P = 100 the difference between classification error and generalization is 2.353 and 2.9248 for SGD and vSGD-l, respectively, while it is only 0.4772 and 0.4416 for BFGS and l-BFGS. This observation is somewhat surprising since it is widely believed that stochastic optimization methods generalize better than second-order batch methods. Note that for larger datasets the difference be-



(a) P = 50 (b) P = 100

(c) P = 200 (d) P = 500

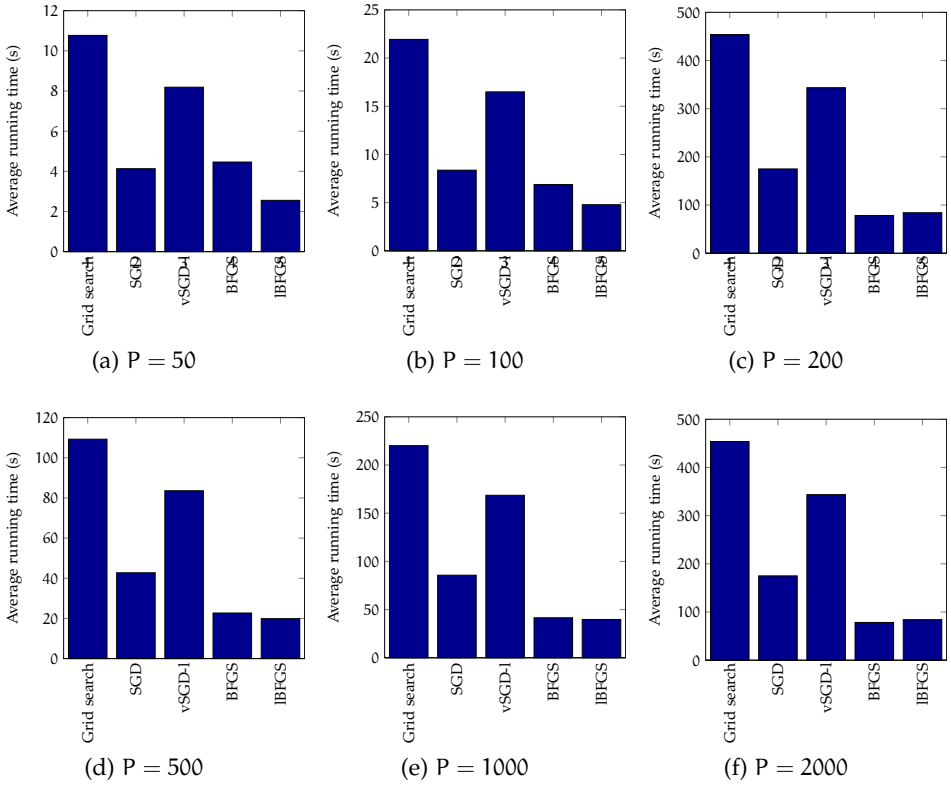
(e) P = 1000 (f) P = 2000

Figure 15: Average learning curve for datasets of size P.

tween training error and generalization error vanishes. Actually, for large P we sometimes observe that the generalization error is better than the training error. In these cases, we were lucky in drawing 'easy' test sets.

4.2.3.3 *Running time*

Fig. 16 provides the average running time for the optimization methods. From the histograms we observe that the relative computational cost does not change whenever we increase the size of the dataset. The lowest running time is achieved by l-BFGS. For small datasets



(a) P = 50 (b) P = 100 (c) P = 200

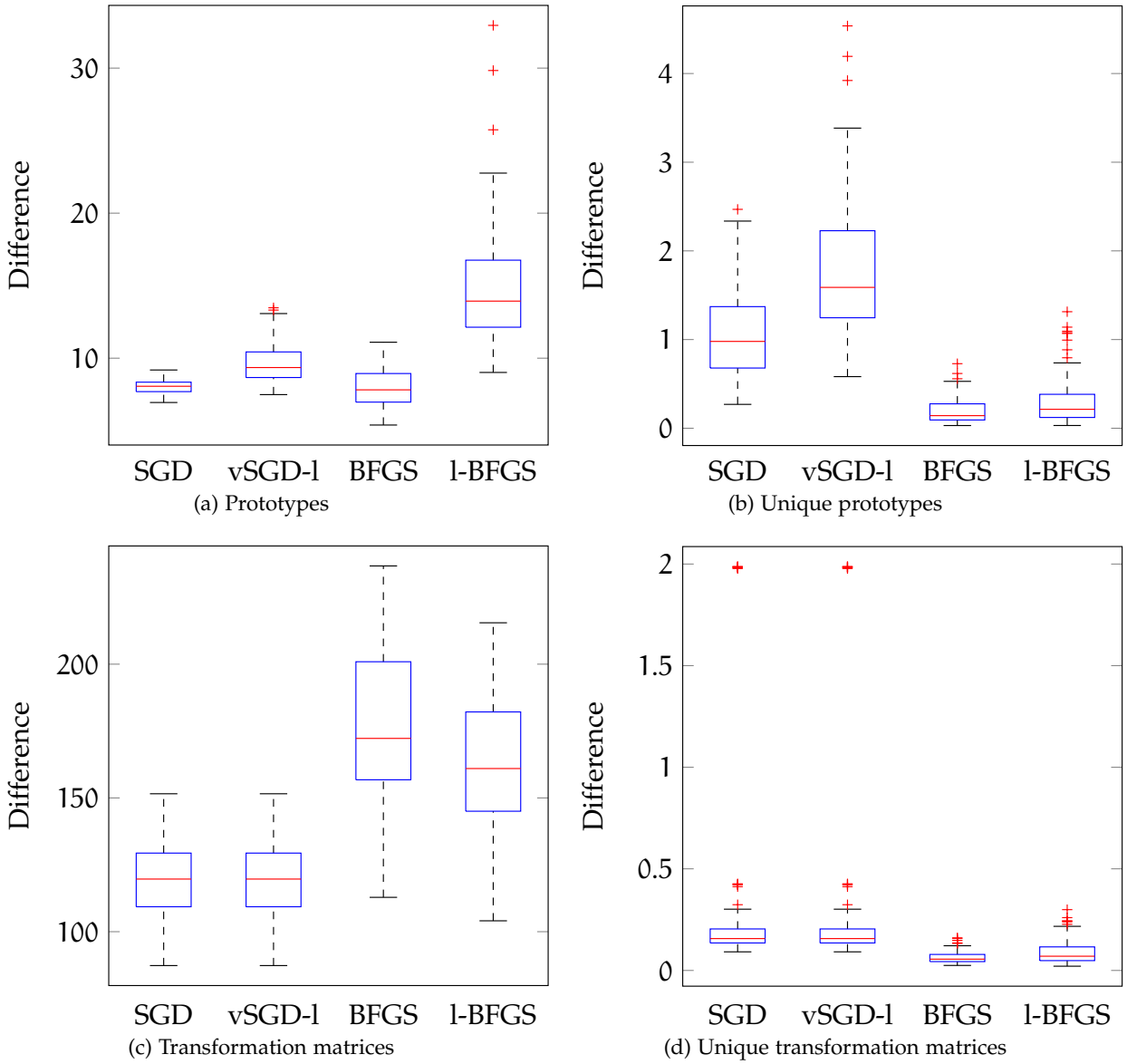
(d) P = 500 (e) P = 1000 (f) P = 2000

Figure 16: Histograms of the average runtime for datasets containing P examples.

it has slightly lower running time than BFGS, while it has similar running time for larger datasets. The computational cost of vSGD-l is approximately four times more than the second-order batch methods, and about two times more than plain SGD (due to double gradient evaluations). Nevertheless, the plain SGD is computationally more expensive because it requires a grid search for the learning rates, which is the most expensive part of all optimization. In total, the plain SGD is thus approximately two times as expensive as vSGD-l, and about 8 times as expensive as (l-)BFGS.

4.2.4 *Uniqueness of solution*

In this chapter we employed the following optimization strategy for GMLVQ. We removed the constraints from the optimization problem, and used solvers that are appropriate for unconstrained optimization. As thoroughly discussed in section 3.3, this strategy is possible because we can project to a unique solution in a single step (that does not affect the cost function value). In this section we verify that this projection results in a unique solution. We decide to use the results from datasets of dimension N = 50. For each of the 10 datasets, we have evaluated 5 runs of the optimization methods for different pa-



(a) Prototypes

(b) Unique prototypes

(c) Transformation matrices

(d) Unique transformation matrices

Figure 17: Box plots of the pairwise differences of the obtained prototypes and transformation matrices for different parameter initializations. The raw differences are shown at the left in a) and b), while b) and d) display the differences for the prototypes and transformation matrix after projection.

rameter initializations. We expect that each run obtains a similar solution, despite its different starting point. That is to say, for each run we find a local minimum of the same cost function value. However, without a projection step the obtained prototypes and transformation matrix might be very different. In the following we first consider the naïve approach by comparing the pairwise differences between the obtained solutions in the different runs. Note that 5 runs results in 10 pairwise differences, for each dataset. The differences are mea-

sured in terms of the l^2 norm for the prototypes, and the Frobenius norm for the transformation matrices. Fig. 17 a), c) shows box plots of the pairwise differences of the prototypes and transformation matrix for SGD, vSGd-l, BFGS and l-BFGS. From the plots we observe that the difference for the prototypes and transformation matrix is quite large for all optimization methods. However, the difference is slightly larger for the second-order batch methods.

Fig. 17 b), d) shows box plots of the pairwise differences of the prototypes and transformation matrix *after projection to the feasible set*. We refer to section 3.2 for more details about the projections that aim to select a unique solution. Note that we do not need to project the rows of Ω into the span of the data, since the generated data sets are of full rank. From the box plots we see that the difference of the unique prototypes and transformation matrix are much smaller than the differences for the raw prototypes and transformation matrices. Nevertheless, these differences depend on the used optimization method. Stochastic optimization methods have much larger differences than the second-order batch methods. These findings confirm that the stochastic optimization methods are worse optimizers than the second-order batch methods. Namely, for different initialization the difference between the found solutions is larger. Note that the difference is somewhat larger for prototypes than for transformation matrix. Therefore, it is likely that the GMLVQ cost function is more sensitive to a change in transformation matrix than a change in the prototype variables. In general, the differences are rather small, and we conclude that we obtain the same solution from different parameter initializations.

4.3 CONCLUSION

In this chapter we have evaluated the performance of SGD, adaptive learning rates for SGD, BFGS and l-BFGS on artificially generated GMLVQ problems. The main finding is that (limited) BFGS outclass the stochastic optimization methods. We demonstrated that BFGS should be the method of choice for low-dimensional datasets (up to 25 dimensions), while limited memory BFGS is the best choice in terms of generalization behavior and running time for high dimensional datasets. Interestingly, the difference in performance between the (l-)BFGS and the stochastic methods is large for small datasets, and seems to vanish whenever the size of the dataset increases.

The adaptive learning rates for SGD, especially the block-specific vSGD-b1 and vSGD-b2 variants, were too conservative to show good performance. The local variant, vSGD-l, performed only slightly worse than plain SGD, but is still too conservative. We think, however, that adaptive learning rates for SGD are an interesting line of research,

and that future work might increase the effectiveness of these methods for large-scale learning problems.

Another interesting result of the experiments is that we did not experience over fitting behavior. That is to say, optimization methods that showed better performance on the training set also showed better performance on the test set. We considered, however, only one prototype per class, and adding more prototypes might result in over fitting. Further studies might thus investigate the role of number of prototypes on the performance of the optimization methods.

Finally, the stationarity conditions presented in Chapter 3 suggest that the sub-Hessian of both the prototypes and transformation matrix has several zero eigenvalues. It might be interesting to investigate whether (l-)BFGS can deal with such curvature properties.

A

APPENDIX

A.1 DERIVATION OF GRADIENT TERMS

In this section we present the derivatives of the cost function of GMLVQ, defined in Eq. 11, with respect to a single prototype $\vec{w}^j$ and the transformation matrix Ω. In the following we only present the single example derivatives since the derivative of the sum is equal to the sum of derivatives.

A.1.1 *Prototype gradient*

By consistently applying the chain rule and product rule we obtain the derivative of the single example contribution $\varphi(e^\mu)$ with respect to a single prototype $\vec{w}^j$:

$$\frac{\partial \varphi(e^\mu)}{\partial \vec{w}^j} = \varphi'(e^\mu)\,\frac{\partial e^\mu}{\partial \vec{w}^j} \tag{97}$$

$$= \varphi'(e^\mu)\left[\frac{\partial e^\mu}{\partial d_+^\mu}\,\frac{\partial d_+^\mu}{\partial \vec{w}^j} + \frac{\partial e^\mu}{\partial d_-^\mu}\,\frac{\partial d_-^\mu}{\partial \vec{w}^j}\right] \tag{98}$$

In the following we present the missing partial derivatives in Eq. 108. The derivative of the logistic function reads as

$$\varphi'(e^\mu) = \gamma\,\varphi(e^\mu)\,(1-\varphi(e^\mu))\,. \tag{99}$$

With some effort we obtain the derivatives of e^μ with respect to closest correct and closest incorrect distance

$$\frac{\partial e^\mu}{\partial d_+^\mu} = \frac{2d_-^\mu}{(d_+^\mu + d_-^\mu)^2}, \tag{100}$$

$$\frac{\partial e^\mu}{\partial d_+^\mu} = \frac{-2d_+^\mu}{(d_+^\mu + d_-^\mu)^2}, \tag{101}$$

respectively. Now, recall that $d_\pm^\mu = \sum_k \Psi_k^{\mu\pm} d^\Omega(\vec{x}^\mu, \vec{w}_k)$ is a sum over the prototypes that single out the distance to the closest correct or incorrect prototype. Hence, the derivative can be rewritten to

$$\frac{\partial d_\pm^\mu}{\partial \vec{w}^j} = \sum_k \Psi_k^{\mu\pm}\,\frac{d^\Omega(\vec{x}^\mu, \vec{w}_k)}{\partial \vec{w}^j} \tag{102}$$

$$= \sum_k \Psi_k^{\mu\pm}\,\frac{d^\Omega(\vec{x}^\mu, \vec{w}_k)}{\partial \vec{w}_k}. \tag{103}$$

The derivative of d^{Ω} with respect to $\vec{w}^j$ follows from Eq. 77 of the Matrix Cookbook [27], and plugging it in gives us

$$\frac{\partial d_{+}^{\mu}}{\partial \vec{w}^j} = \sum_k -\Psi_k^{\mu\pm} 2\Lambda(\vec{x}^{\mu} - \vec{w}_k). \tag{104}$$

Putting all partial derivatives together, we obtain the final derivative

$$\frac{\partial \varphi(e^{\mu})}{\partial \vec{w}^j} = \chi_j^{\mu} \Lambda(\vec{x}^{\mu} - \vec{w}^j) \tag{105}$$

where χ_j^{μ} depends on the actual role of the role of the prototype $\vec{w}^j$:

CLOSEST CORRECT PROTOTYPE

$$\begin{aligned}\chi_j^{\mu} &= -4\gamma\, \varphi'(e^{\mu}) \frac{d_{-}^{\mu}}{(d_{+}^{\mu} + d_{-}^{\mu})^2} \Lambda(\vec{x}^{\mu} - \vec{w}^j) \\ &= -4\gamma\, \varphi(e^{\mu})(1 - \varphi(e^{\mu})) \frac{d_{-}^{\mu}}{(d_{+}^{\mu} + d_{-}^{\mu})^2} \Lambda(\vec{x}^{\mu} - \vec{w}^j)\end{aligned} \tag{106}$$

CLOSEST INCORRECT PROTOTYPE

$$\begin{aligned}\chi_j^{\mu} &= 4\gamma\, \varphi'(e^{\mu}) \frac{d_{+}^{\mu}}{(d_{+}^{\mu} + d_{-}^{\mu})^2} \Lambda(\vec{x}^{\mu} - \vec{w}^j) \\ &= 4\gamma\, \varphi(e^{\mu})(1 - \varphi(\gamma e^{\mu})) \frac{d_{+}^{\mu}}{(d_{+}^{\mu} + d_{-}^{\mu})^2} \Lambda(\vec{x}^{\mu} - \vec{w}^j)\end{aligned} \tag{107}$$

OTHERWISE $\chi_j^{\mu} = 0$

A.1.2 *Transformation matrix gradient*

Analogous to the prototype gradient, we present the derivative of the single example contribution with respect to the transformation matrix Ω

$$\frac{\partial \varphi(\gamma e^{\mu})}{\partial \Omega} = \varphi'(\gamma e^{\mu}) \left[\frac{\partial e^{\mu}}{\partial d_{+}^{\mu}} \frac{\partial d_{+}^{\mu}}{\partial \Omega} + \frac{\partial e^{\mu}}{\partial d_{-}^{\mu}} \frac{\partial d_{-}^{\mu}}{\partial \Omega} \right]. \tag{108}$$

The derivatives $\varphi'(\gamma e^{\mu})$, $\frac{\partial e^{\mu}}{\partial d_{+}^{\mu}}$ and $\frac{\partial e^{\mu}}{\partial d_{-}^{\mu}}$ are already presented in Eq. 99, 100, 101. The only missing derivative is

$$\frac{\partial d_{\pm}^{\mu}}{\partial \Omega} = \sum_j \Psi_k^{\mu\pm} \frac{d^{\Omega}(\vec{x}^{\mu}, \vec{w}^j)}{\partial \Omega} \tag{109}$$

where the partial derivative of distance measure with respect to Ω

$$\frac{\partial d^{\Omega}(\vec{x}^{\mu}, \vec{w}^j)}{\partial \Omega} = \Omega(\vec{x}^{\mu} - \vec{w}^j)(\vec{x}^{\mu} - \vec{w}^j)^{\top} \tag{110}$$

follows from Eq. 69 of the Matrix Cookbook [27]. Now, putting the partial derivatives together results in the final gradient term

$$\frac{\partial \varphi(\gamma e^{\mu})}{\partial \Omega} = \sum_{j} \chi_j^{\mu} \Omega(\vec{x}^{\mu} - \vec{w}^{j})(\vec{x}^{\mu} - \vec{w}^{j})^{\top} \tag{111}$$

$$= \Omega \sum_{j} \chi_j^{\mu} (\vec{x}^{\mu} - \vec{w}^{j})(\vec{x}^{\mu} - \vec{w}^{j})^{\top} \tag{112}$$

where χ_j^{μ} depends on the actual role of the prototype which is defined in Eq. 106, 107.

A.2 DERIVATION OF HESSIAN MATRIX

We assume that the Hessian matrix is block diagonal, and thus only consider the second-order derivatives with respect to the same group of variables.

A.2.1 *Prototypes*

We first present the second-order derivatives with respect to prototype variables $\vec{w}_l^j$ and $\vec{w}_p^k$:

$$\frac{\partial^2 \varphi(e^{\mu})}{\partial \vec{w}_l^j \partial \vec{w}_p^k} = \varphi''(e^{\mu}) \frac{\partial e^{\mu}}{\partial \vec{w}^k} \frac{\partial e^{\mu}}{\partial \vec{w}_l^j} + \varphi'(e^{\mu}) \frac{\partial^2 e^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^k} \tag{113}$$

where

$$\begin{aligned} \varphi''(e^{\mu}) =& \frac{-\gamma^2 \exp(\gamma e^{\mu})(\exp(\gamma e^{\mu}) - 1)}{(\exp(\gamma e^{\mu}) + 1)^3} \\ =& \gamma^2 \varphi(e^{\mu}) \left((1 - \varphi(e^{\mu}))^2 - \varphi(e^{\mu})(1 - \varphi(e^{\mu})) \right) \end{aligned} \tag{114}$$

and the derivatives $\frac{\partial e^{\mu}}{\vec{w}_p^k}$ and $\varphi'(e^{\mu})$ are already given in section A.1.1. For clarity purposes, we immediately distinguish between the roles of prototypes $\vec{w}^j$ and $\vec{w}^k$ for the derivative $\frac{\partial^2 e^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^k}$. If $\vec{w}^j$ and $\vec{w}^k$ are the *closest correct prototype* (and thus the same prototype), then the derivative reads as:

$$\frac{\partial^2 e^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^k} = \frac{\partial^2 e^{\mu}}{\partial^2 d_+^{\mu}} \frac{\partial d_+^{\mu}}{\partial \vec{w}^k} \frac{\partial d_+^{\mu}}{\partial \vec{w}_l^j} + \frac{\partial e^{\mu}}{\partial d_+^{\mu}} \frac{\partial^2 d_+^{\mu}}{\partial \vec{w}_l^j \partial}. \tag{115}$$

If $\vec{w}^j$ and $\vec{w}^k$ are both the *closest wrong prototype* (and thus the same prototype), then we obtain the derivative:

$$\frac{\partial^2 e^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^k} = \frac{\partial^2 e^{\mu}}{\partial^2 d_-^{\mu}} \frac{\partial d_-^{\mu}}{\partial \vec{w}_p^k} \frac{\partial d_-^{\mu}}{\partial \vec{w}_l^j} + \frac{\partial e^{\mu}}{\partial d_-^{\mu}} \frac{\partial^2 d_-^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^j}. \tag{116}$$

If $\vec{w}^j$ and $\vec{w}^k$ are the *closest correct and closest wrong prototype* (or vice versa by symmetry of second derivatives), then the second-order derivative is given by:

$$\frac{\partial^2 e^{\mu}}{\partial \vec{w}_l^j \partial \vec{w}_p^k} = \frac{\partial^2 e^{\mu}}{\partial d_+^{\mu} \partial d_-^{\mu}} \frac{\partial d_-^{\mu}}{\partial \vec{w}_p^k} \frac{\partial d_+^{\mu}}{\partial \vec{w}_l^j} \tag{117}$$

In all other cases, that is, if either $\vec{w}^j$ or $\vec{w}^k$ is not a winning prototype then the second derivative will be zero. In the following we present the missing partial derivatives. The second derivatives of the distance measure with respect to the prototypes reads as:

$$\frac{\partial^2 e^\mu}{\partial^2 d_+^\mu} = \frac{-4 d_-^\mu}{(d_+^\mu + d_-^\mu)^3} \tag{118}$$

$$\frac{\partial^2 e^\mu}{\partial d_+^\mu \partial d_-^\mu} = \frac{2(d_+^\mu - d_-^\mu)}{(d_+^\mu + d_-^\mu)^3} \tag{119}$$

$$\frac{\partial^2 e^\mu}{\partial^2 d_+^\mu} = \frac{4 d_+^\mu}{(d_+^\mu + d_-^\mu)^3} \tag{120}$$

The second derivative of the distance measure to a prototype $\vec{w}_l^j$:

$$\frac{\partial^2 d_\pm^\mu}{\partial \vec{w}_l^j \partial \vec{w}_p^k} = \Lambda_{lp} \tag{121}$$

Now, all partial derivatives should be plugged into Eq. 113 to obtain the sub Hessian matrix of the prototype variables.

A.2.2 *Transformation matrix*

We present second-order derivatives with respect to the transformation matrices Ω_{lm} and Ω_{np}:

$$\frac{\partial^2 \varphi(e^\mu)}{\partial \Omega_{lm} \partial \Omega_{np}} = \varphi''(e^\mu) \frac{\partial e^\mu}{\partial \Omega_{lm}} \frac{\partial e^\mu}{\partial \Omega_{np}} + \varphi'(e^\mu) \frac{\partial^2 e^\mu}{\partial \Omega_{lm} \partial \Omega_{np}} \tag{122}$$

where $\varphi''(e^\mu)$ is defined in Eq. 114, and the first order derivatives with respect to Ω were obtained in section A.1.2. The second-order derivative of e^μ with respect to the transformation matrix elements result in a complex chain of partial derivatives:

$$\begin{aligned} \frac{\partial^2 e^\mu}{\partial \Omega_{lm} \partial \Omega_{np}} = & \frac{\partial^2 e^\mu}{\partial \Omega_{np} \partial d_+^\mu} \frac{\partial d_+^\mu}{\partial \Omega_{lm}} + \frac{\partial e^\mu}{\partial d_+^\mu} \frac{\partial^2 d_+^\mu}{\partial \Omega_{lm} \partial \Omega_{np}} \\ & + \frac{\partial^2 e^\mu}{\partial \Omega_{np} \partial d_-^\mu} \frac{\partial d_-^\mu}{\partial \Omega_{lm}} + \frac{\partial e^\mu}{\partial d_-^\mu} \frac{\partial^2 d_-^\mu}{\partial \Omega_{lm} \partial \Omega_{np}} \end{aligned} \tag{123}$$

In the following we only present the missing derivatives in Eq. 123. We first first present the derivative:

$$\frac{\partial^2 e^\mu}{\partial d_+^\mu \partial \Omega_{np}} = \frac{\partial^2 e^\mu}{\partial^2 d_+^\mu} \frac{\partial d_+^\mu}{\partial \Omega_{np}} + \frac{\partial^2 e^\mu}{\partial d_+^\mu \partial d_-^\mu} \frac{\partial d_-^\mu}{\partial \Omega_{np}} \tag{124}$$

$$\frac{\partial^2 e^\mu}{\partial d_-^\mu \partial \Omega_{np}} = \frac{\partial^2 e^\mu}{\partial^2 d_-^\mu} \frac{\partial d_-^\mu}{\partial \Omega_{np}} + \frac{\partial^2 e^\mu}{\partial d_+^\mu \partial d_-^\mu} \frac{\partial d_+^\mu}{\partial \Omega_{np}} \tag{125}$$

(126)

By Eq. 66 of the Matrix Cookbook [27], we obtain the second-order derivative of the distance measure with respect to transformation matrix:

$$\frac{\partial d_+^\mu}{\partial \Omega_{lm} \partial \Omega_{np}} = \sum_j \Psi_j^{\mu+} \left(J^{np} (x^\mu - \vec{w}^j)(x^\mu - \vec{w}^j)^\top \right)_{lm}, \tag{127}$$

where J is the single entry matrix. The derived partial derivatives should be plugged into Eq. 122.

BIBLIOGRAPHY

[1] Ethem Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.

[2] Sue Becker and Yann Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37. San Matteo, CA: Morgan Kaufmann, 1988.

[3] M Biehl, Barbara Hammer, Frank-Michael Schleif, P Schneider, and Thomas Villmann. Stationarity of matrix relevance learning vector quantization. *Machine Learning Reports*, 3:1–17, 2009.

[4] Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *The Journal of Machine Learning Research*, 10:1737–1754, 2009.

[5] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL http://leon.bottou.org/papers/bottou-98x. revised, oct 2012.

[6] Léon Bottou and Olivier Bousquet. 13 the tradeoffs of large-scale learning. *Optimization for Machine Learning*, page 351, 2011.

[7] Léon Bottou and Yann LeCun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. URL http://leon.bottou.org/papers/bottou-lecun-2004.

[8] Stephen Boyd and Almir Mutapcic. Stochastic subgradient methods. *Notes for EE364b, Stanford, CA: Stanford Univ*, 2008.

[9] Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. *lecture notes of EE3920, Stanford University, Autumn Quarter*, 2004, 2003.

[10] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[12] Kerstin Bunte, Petra Schneider, Barbara Hammer, Frank-Michael Schleif, Thomas Villmann, and Michael Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26:159–173, 2012.

[13] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[14] Koby Crammer, Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Margin analysis of the lvq algorithm. *Advances in neural information processing systems*, 15:462–469, 2002.

[15] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley-interscience, 2012.

[16] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[17] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[18] Barbara Hammer and Thomas Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15(8):1059–1068, 2002.

[19] Barbara Hammer, Marc Strickert, and Thomas Villmann. On the generalization ability of grlvq networks. *Neural Processing Letters*, 21(2):109–120, 2005.

[20] Barbara Hammer, Marc Strickert, and Thomas Villmann. Supervised neural gas with general similarity measure. *Neural Processing Letters*, 21(1):21–44, 2005.

[21] T. Kohonen. *Learning Vector Quantization for Pattern Recognition*. Report TKK-F-A. Helsinki University of Technology, 1986. ISBN 9789517539500. URL http://books.google.nl/books?id=PwEkAAAACAAJ.

[22] Teuvo Kohonen. Improved versions of learning vector quantization. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 545–550. IEEE, 1990.

[23] Nicolas Le Roux, Yoshua Bengio, and Andrew Fitzgibbon. 15 improving first and second-order methods by modeling uncertainty. *Optimization for Machine Learning*, page 403, 2011.

[24] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[25] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Andrew Ng, and Quoc V Le. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[26] Jorge. Nocedal and Stephen J Wright. *Numerical optimization*, volume 2. Springer New York, 1999.

[27] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook, 2008. *URL http://www2. imm. dtu. dk/pubdb/p. php*, 3274, 2008.

[28] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. *Advances in neural information processing systems*, pages 423–429, 1996.

[29] Tom Schaul and Yann LeCun. Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients. *arXiv preprint arXiv:1301.3764*, 2013.

[30] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *arXiv preprint arXiv:1206.1106*, 2012.

[31] Petra Schneider, Michael Biehl, and Barbara Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009.

[32] Nicol Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. *aa*, 2007.

[33] Sambu Seo and Klaus Obermayer. Soft learning vector quantization. *Neural computation*, 15(7):1589–1604, 2003.

[34] Sambu Seo, Mathias Bode, and Klaus Obermayer. Soft nearest prototype classification. *Neural Networks, IEEE Transactions on*, 14 (2):390–398, 2003.

[35] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[36] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for Machine Learning*. Mit Pr, 2012.

[37] M. Strickert. Enhancing m|g|rlvq by quasi step discriminatory functions using 2nd order training. In *Machine Learning Reports 06/2011*, pages XX–YY, 2011. ISSN:1865-3960 http://www.techfak.uni-bielefeld.de/fschleif/mlr/mlr_06_2011.pdf.

[38] M. Strickert, B. Hammer, T. Villmann, and M. Biehl. Regularization and improved interpretation of linear data mappings and adaptive distance measures. In *Proc. IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2013. In press.

[39] Vladimir Vapnik. *The nature of statistical learning theory*. springer, 1999.

[40] Thomas Villmann and Barbara Hammer. Theoretical aspects of kernel glvq with differentiable kernel. *ICOLE 2009, Lessach, Austria*, page 133, 2009.