DISSERTATION

submitted to the

Combined Faculty for the Natural Sciences and Mathematics

of

Heidelberg University, Germany

for the degree of
Doctor of Natural Sciences

put forward by
M.Sc. Vera Trajkovska
born in Kumanovo

Date of oral examination: .....

Learning Probabilistic Graphical Models for Image Segmentation

Advisors: Prof. Dr. Christoph Schnörr
Prof. Dr. Björn Ommer

Zusammenfassung

Probabilistische grafische Modelle bieten ein mächtiges Framework für die Darstellung von Strukturen in Bildern. Auf diesem Konzept basierend wurden viele Inferenz- und Lernalgorithmen entwickelt. Jedoch, sowohl Lernen als auch Inferenz sind im allgemeinen Fall kombinatorische NP-harte Probleme. Infolgedessen wurden Relaxierungsmethoden entwickelt um die Originalprobleme zu approximieren aber mit dem Vorteil, dass sie vom Rechenaufwandt effizient sind. In dieser Arbeit betrachten wir das Lernproblem von binären graphischen Modellen und deren Relaxierungen. Zwei neuartige Methoden zum Bestimmen der Modellparameter von diskreten Energiefunktionen aus Trainingsdaten werden vorgeschlagen. Das Lernen der Modellparameter löst elegant das Problem, diese heuristisch zu bestimmen.

Motiviert durch gängige Lernmethoden, die auf die Minimierung des Trainingsfehlers oder einer Verlustfunktion aufbauen, entwickeln wir eine neue Lernmethode ähnlich zu structural SVM. Jedoch von dem Gesichtspunkt der Optimierung ist diese einfacher. Wir benennen diese Methode als linearisierten Ansatz (LA), da er auf linear abhängige Potentiale beschränkt ist. Die Linearität von LA ist entscheidend um eine feste konvexe Relaxierung zu erhalten, die den Einsatz von Standard-Inferenz-Lösern zum behandeln der Teilprobleme ermöglicht die beim Lösen des Gesamtproblems auftreten.

Allerdings wird diese Art von Lernmethoden fast nie die optimale Lösung liefern oder perfekte Performance auf der ganzen Trainingsmenge. Was würde also passieren, wenn das gelernte grafische Modell auf den ganzen Trainingsdaten die exakte Ground-Truth-Segmentierung wiedergeben würde? Würde das einen Vorteil für die Vorhersage versprechen?

Inspiriert von inverser Optimierung nutzen wir die inversen linearen Programmierung um eine weitere Lernmethode zu entwickeln, bezeichnet als inversen linearen Programmierungsansatz (invLPA). Dieser Ansatz verbessert die trainierten graphischen Modelle weiter unter Verwendung der zuvor eingeführten Methoden und ist fähig die exakten Ground-Truth-Daten auf der Trainingsmenge vorherzusagen. Die empirischen Ergebnisse aus der Umsetzung von invLPA geben Antworten auf unsere zuvor gestellte Frage.

LA ist in der Lage gleichzeitig die unären und paarweisen Potentiale zu lernen während dies mit invLPA wegen der gewählten Repräsentierung nicht möglich ist. Auf der anderen Seite jedoch ist invLPA bezüglich der verwendeten Fittingmethode sehr flexibel, da die Potentiale nicht auf eine feste Gestalt beschränkt sind.

Auch wenn die korrigierten Potentiale mit invLPA immer zu Ground-Truth- Segmentierung der Trainingsdaten führen, kann invLPA nur Korrekturen der Vordergrundsegmente vornehmen. Wegen der relaxierten Problemformulierung beeinflußt dies jedoch nicht das endgültige Ergebnis der Segmentierung. Vielmehr, solange

wir mit vorberechneten Modellparametern einer hinreichend guten Lernmethode initialisieren, wird diese Schwäche von invLPA das endgültige Vorhersageergebnis nicht signifikant beeinflussen.

Die Performance von unseren vorgeschlagenen Lernmethoden wird sowohl auf synthetischen als auch auf realen Datensätzen evaluiert. Wir zeigen das LA konkurrenzfähig ist im Vergleich zu anderen Lernmethoden, die Verlustfunktionen verwenden basierend auf Maximum a Posteriori Marginal (MPM) und Maximum Likelihood Estimation (MLE). Darüber hinaus veranschaulichen wir die Vorteile des Lernens mit Hilfe von inverser linearer Programmierung. In einem weiteren Experiment demonstrieren wir die Vielseitigkeit von unseren Lernansätzen indem wir LA anwenden zum Lernen von Bewegungssegmentierung in Videosequenzen und vergleichen dies gegenüber state-of-the-art Segmentierungsalgorithmen.

Abstract

Probabilistic graphical models provide a powerful framework for representing image structures. Based on this concept, many inference and learning algorithms have been developed. However, both algorithm classes are NP-hard combinatorial problems in the general case. As a consequence, relaxation methods were developed to approximate the original problems but with the benefit of being computationally efficient. In this work we consider the learning problem on binary graphical models and their relaxations. Two novel methods for determining the model parameters in discrete energy functions from training data are proposed. Learning the model parameters overcomes the problem of heuristically determining them.

Motivated by common learning methods which aim at minimizing the training error measured by a loss function we develop a new learning method similar in fashion to structured SVM. However, computationally more efficient. We term this method as linearized approach (LA) as it is restricted to linearly dependent potentials. The linearity of LA is crucial to come up with a tight convex relaxation, which allows to use off-the-shelf inference solvers to approach subproblems which emerge from solving the overall problem.

However, this type of learning methods almost never yield optimal solutions or perfect performance on the training data set. So what happens if the learned graphical model on the training data would lead to exact ground segmentation? Will this give a benefit when predicting?

Motivated by the idea of inverse optimization, we take advantage of inverse linear programming to develop a learning approach, referred to as inverse linear programming approach (invLPA). It further refines the graphical models trained, using the previously introduced methods and is capable to perfectly predict ground truth on training data. The empirical results from implementing invLPA give answers to our questions posed before.

LA is able to learn both unary and pairwise potentials jointly while with invLPA this is not possible due to the representation we use. On the other hand, invLPA does not rely on a certain form for the potentials and thus is flexible in the choice of the fitting method.

Although the corrected potentials with invLPA always result in ground truth segmentation of the training data, invLPA is able to find corrections on the foreground segments only. Due to the relaxed problem formulation this does not affect the final segmentation result. Moreover, as long as we initialize invLPA with model parameters of a learning method performing sufficiently well, this drawback of invLPA does not significantly affect the final prediction result.

The performance of the proposed learning methods is evaluated on both synthetic and real world datasets. We demonstrate that LA is competitive in comparison

to other parameter learning methods using loss functions based on Maximum a Posteriori Marginal (MPM) and Maximum Likelihood Estimation (MLE). Moreover, we illustrate the benefits of learning with inverse linear programming. In a further experiment we demonstrate the versatility of our learning methods by applying LA to learning motion segmentation in video sequences and comparing it to state-of-the-art segmentation algorithms.

Acknowledgments

I would like to thank many people without which this thesis would not have been possible.

First of all I am very grateful to my supervisor Professor Christoph Schnörr for giving me the opportunity to join the RTG at the Faculty of Mathematics and Computer Science. Under his supervision I was able to benefit from his guidance, support, help and understanding. I am very fortunate, that I had the oppprtunity to have him as a supervisor and all my knowledge during my PhD studies I owe to him.

I would like to thank a lot Stefania without whom as well a lot of this work would not have happened. And of course Bogdan who guided me at the beginning of my PhD work. Many many thanks to Florian, first of all for proofreading this thesis. And for all the discussions, all the help not only for my work but also for my bike repair etc. For letting me share the office with him (and Stefania) for some time and for standing the "warmth" in our office. I am very thankful to Arati and Eva for being my first friends in Heidelberg. Special thanks goes to Barbara for our lunches together, all the time spent together and for always being able to listen to my complaints and problems. Thanks to Evelyn for all the administrative help. I would like to thank Freddie for all the help and discussions. I am thankful to Fabian for helping me optimize my code, swimming together etc. I would also like to thank all my colleagues which were also part of my PhD years: Ecaterina, Artjom, Tabea, Andreas, Francesco, Mattia, Robert, Jan, Leonid, Lucas, Ruben, Fabrizio, Judit, Alexander.

And I would like to express my deepest gratitude to Matthias for all the love, help and support I had from him the last two years. Who always encouraged me and belived in me. Thank you as well for proofreading my thesis. And finally I would like to thank my whole family my father Chedomir, my mother Frosina, my sister Violeta and my brother Simon. Even far away I always had your love and support. And I would have never been where I am today if it would not have been you encouraging me in all my goals.

Funding by the Deutsche Forschungsgemeinschaft via the research training group 1653 Spatio / Temporal Graphical Models and Applications in Image Analysis is gratefully acknowledged.

List of Publications

Part of this thesis has been published at a conference.

V. Trajkovska, P. Swoboda, F. Åström, and S. Petra, *Graphical Model Parameter Learning by Inverse Linear Programming*, LNCS 10302, pp. 323–334, Springer, 2017

Contents

1. Introduction

1.1. Overview and Motivation

Probabilistic graphical models are nowadays widely used tool in computer vision. Representing an image with the help of a graph together with its neighborhood structure allows us to use graphical models and solve a task related to the image, such as segmentation, labeling or denoising. A common way is to formulate an energy function which describes the fitness of a variable configuration, and then to apply numerical optimization to determine an optimal or at least good solution. For instance let us consider the binary image labeling problem: the aim is to assign each pixel either to a foreground or a background set. Fig. 1.1 provides an example where the horse is marked as foreground segment and the remaining pixels as background.

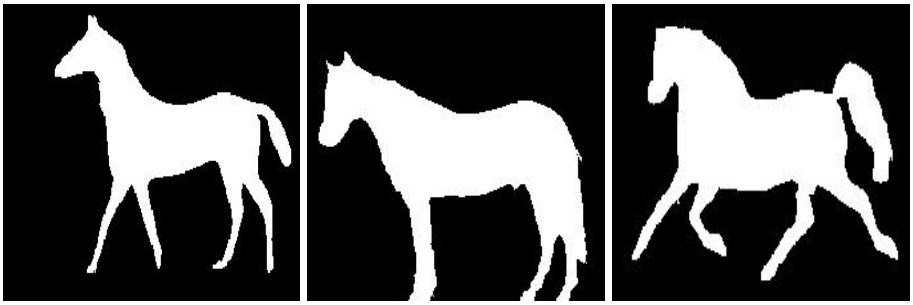
Representing the image labeling problem using graphical models, requires to define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ representing the image pixels or superpixels and edges $\mathcal{E}$ representing neighborhood structures. On this a discrete energy function,

$$E_\theta(x) = \sum_{i \in \mathcal{V}} \theta^i(x_i) + \sum_{e \in \mathcal{E}} \theta^e(x_e). \tag{1.1}$$

is defined and solved using a suitable optimization algorithm.

For (1.1) we need to choose parameters θ which depend on some data feature vectors in the image, which should capture sufficient information of the image structure. However, it is not always straight-forward to choose suitable parameters. Moreover, different choices lead to different results depending on the application at hand. One way is to define the model parameters based on some heuristic. Often this is not a straight-forward task, for instance, when the data has high variances in shape and color as in the Weizmann horse dataset [BU08] visualized in Fig. 1.1. While in the first image the horse can be easily segmented from the background with some segmentation algorithm like a minimum cut, this is not the case for the other two images. The second image is quite difficult to be segmented as the head of the horse is not easy to be distinguished from the background behind it, and the same holds for the horse tail. In addition the whole horse region is not homogeneous with respect to color. In contrast, the third image has a homogeneous color in the horse area, but part of the background shares the same color. In particular this part is connected to the horse region. A simple segmentation algorithm would tend to segment this image as horse together with the white background part or merge the horse head and tail with the background for the second image. Hand tunning of the model parameters for each image separately could lead to some reasonable segmentation results. This is, however, time consuming and is not an appropriate solution especially in the case

(a) input



(b) ground truth segmentation

Figure 1.1. - (a) Three images of horses from the Weizmann horse dataset [BU08] with (b) their ground truth segmentation. While the first image can be segmented with some segmentation algorithm, for instance max-flow algorithms which find the minimum cut, the two other images would certainly fail even when powerful features are used. The reason is variances in the foreground class and the similarity of the foreground and background class for the last two images respectively.

when the data to be processed is huge.

A more appropriate way, is to *learn* the model parameters, given a set of training data with available ground truth segmentation, e.g. obtained by hand-annotating images. For the case when part of our dataset consists from the three images from Fig. 1.1 learning model parameters results in model parameters which equally represent the whole dataset. The data on which the model parameters are trained should be representative for the data on which they are tested. Otherwise the segmentation performance will deteriorate.

Based on the learned parameters new model parameters corresponding to the testing data are predicted using some prediction model, e.g. least-squares, Gaussian regression.

Common approach for learning is to define an objective function, or loss function, which depends on the model parameters and measures the prediction error on the training data. After minimizing the function, e.g. using gradient descent-based methods, the learned model parameters are obtained. Computing the prediction error involves minimizing (1.1). As in general (1.1) is NP-hard a relaxed version is considered. Due to this the loss function minimizes error based on approximations. Or in other words by computing "approximate" gradients. Loss functions of this type collect all the training data and learn set of model parameters that best match all data at once. In this fashion the learned set of model parameters might not be able to always reconstruct ground truth for *each* of the training data.

Part of our work is motivated by the idea of inverse linear programming which corrects a given feasible sub-optimal solution to an optimal one. Our first learning method based on this concept, first corrects the learned approximate gradients computed by a learning method such that it leads to the exact ground truth solution for each of the data in the training set. We will refer to this method as inverse linear programming approach (invLPA). Subsequently any existing model parameter prediction method can be applied to compute a prediction on the testing data.

A further part of this work is dedicated to our second novel objective function for learning, referred as linearly parametrized joint learning approach (LA), which has an objective which is a loss function of the common type as explained above.

1.2. Related Work

The literature of learning in general is vast and and we refer the reader to [NL11] for an excellent overview. Important related work on learning using relaxed inference is the work of Wainwright et al. [Wai06] where it was proven that learning can even benefit from approximations as long as the same approximate inference method is used while learning and predicting. In practice the error introduced while learning approximate parameters is partly compensated by the error of the inexact inference.

The literature on inverse linear programming [ZL96, ZL99, AO01] are the basis for the main novel learning method we present in this work. The authors in [ZL96] apply inverse optimization to the minimum cost flow problem and the assignment problem, which result in a linear program. We follow the same idea and develop

an inverse linear program for the relaxed discrete binary energy function for image labeling.

Most learning methods require the definition of a loss function which steers optimization towards a very small training error. A well known learning method based on such a loss function is structured Support Vector Machine (SVM) [FJ08, THJA04, TJHA05], a generalization of the classification SVM, and can be applied to different type of data such as lattices, sets and strings. The basic idea is to minimize a problem specific loss function while maximizing the minimal prediction error on the training data.

One of our proposed learning methods leads to a similar objective as the structured SVM. However, our objective is easier to optimize as we fix one of the variables which has a similar role in the structured SVM learning. In addition we use a different optimization method as compared to structured SVM. We use an enhanced subgradient method which involves simpler numerics than a cutting plane method, as used for structured SVM.

Our subgradient method is also known by the name deflected subgradient method. This subgradient method was first proposed in [CFM75] with the aim speeding up the usually slow subgradient method. The PhD work of [Gut03] more thoroughly analyzes the deflected subgradient method proposed in [CFM75]. The authors in [dF09] discuss two approaches for speeding up the subgradient method which tends to advance in zig zag curves. The deflected subgradient method usually comes with a (modified) Polyak step size first proposed by Polyak in [Pol69]. The Polyak step size acquires knowledge of the optimal objective value which is in general not known. Due to this Polyak proposed a modification of this step size when an upper or lower bound of the function is available.

In order to show competitiveness of learning methods we compare our learning method LA based on loss minimizing to two methods from [Dom13], which compares learning methods with different loss functions. Domke compares loss functions based on Maximum a Posteriori Marginal (MPM) to those based on Maximum Likelihood Estimates (MLE) for the learning problem. However, he uses only inference methods based on MPM. Along with a new (heuristic) optimization method he concludes that MPM loss functions lead to better performance than those based on MLE.

Concerning the real world data we use for evaluation of our learning methods we use the challenging Weizmann horse dataset [BU08]. In addition we use the densely annotated video sequence dataset introduced in [PPTM+16].

1.3. Contribution

The main contribution of this work is two novel methods for learning parameters in graphical models.

In addition we propose a new optimization method and apply it to two metric learning approaches which arise from learning suitable distances with aim improving k-means clustering. Metric learning on the other hand can be considered as learning the unary part of the graphical model. We illustrate the proposed method on few

small datasets and demonstrate its efficiency with comparison to established solvers for semi-definite programming.

The first proposed learning method, inverse linear programming approach (invLPA), we develop, is using the concept of inverse linear programming. This method corrects the "approximate" gradients of another learning method. We show that the corrected potentials correspond to exact ground truth segmentation.

The second proposed method we develop, the linearized approach (LA) shares some common ideas with structured SVM [FJ08, THJA04, TJHA05] but requires less complex numerics procedure for minimizing the loss function during training. We show how to choose a parameter set by hand which enforces uniqueness. Furthermore, we prove convergence of the deflected subgradient method with modified Polyak step size we use for optimizing the objective of LA.

We evaluate our two learning methods both on synthetic and real world datasets and compare to state-of-the-art classification methods. Furthermore, we illustrate the benefit of learning using an objective function where both a regularizer is present as opposed to training a classifier and adding a regularizer in a post processing step. We extend our methods to learn pairwise potentials and demonstrate that they outperform standard regularizers, e.g. the Ising regularizer, when difficult structures have to be learned. Furthermore, jointly learning unary and pairwise potentials with LA is compared to two other learning methods for parameters in graphical models from [Dom13]. In addition to single images we also consider videos and extend LA to motion segmentation learning and provide experiments on the real world dataset [PPTM+16].

Finally, we discuss the benefits as well as the drawbacks from both learning approaches and propose some possible extensions.

1.4. Organization

We organize this work as follows.

In Chapter 2 we introduce all the basic background we use throughout this thesis. We start with definitions from graph theory used in image analysis and proceed with probabilistic graphical models. Next, we present the basic tools from convex optimization in order to proceed with exponential families. The following two sections are dedicated to inference and learning, which are the central tools for solving graphical models. The last section addresses inverse linear programming which is the basis of one of our learning methods presented in Chapter 4.

In Chapter 3 we introduce metric learning methods. We give an overview on the most popular approaches and the utilized numerical techniques. In addition, we propose a new optimization procedure which we also apply to two metric learning objectives. We implement our proposed optimization method to few small datasets and compare it to established semi-definite solvers used for metric learning.

In Chapter 4 we develop our two novel methods which can learn both unary and pairwise potentials in a graphical model. We discuss the proposed methods from a theoretical point of view. Also the optimization methods used in LA are

addressed including proofs for convergence. The differences and similarities of LA and structured SVM are discussed. In our experiments we apply invLPA and LA to synthetic and real-world data sets for image segmentation and quantitatively evaluate invLPA with respect to ground truth. Furthermore, we extend the evaluation of LA to semi-supervised online learning in video sequences.

In our last Chapter 5 we conclude and propose possible extensions of our contribution as further work.

1.5. Notation

The following table provides an overview on the notation used throughout the thesis. Each is introduced in detail in the respective chapter.

Background

$\mathcal{G}$	a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ which is a pair of a set of nodes (vertices) $\mathcal{V}$, and a set of edges $\mathcal{E}$
X	a random variable, a random vector
x	a value taken by a random variable X
$\mathcal{X}$	all possible events of a random variable X, domain of the random variable X
$p(x)$	a probability distribution of the random variable X
$(X, \mathcal{G})$	a probabilistic graphical model, a pair of a random variable X and a graph $\mathcal{G}$
X_i	a random variable corresponding to the node $i \in \mathcal{V}$ from the set of nodes $\mathcal{V}$ in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and taking values in $\mathcal{X}_i$
X_A	$:= (X_i)_{i \in A}$ a random sub-vector of X consisting of the random variables X_i corresponding to the nodes of the set $A \subset \mathcal{V}$
ij	an edge from the edge set $\mathcal{E}$, such that $i, j \in \mathcal{V}$
C	a clique of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ which is a subset of the set of vertices $\mathcal{V}$
$\mathcal{C}(\mathcal{G})$	set of maximal cliques of the graph $\mathcal{G}$
$\mathcal{P}(\mathcal{V})$	set of all subsets of $\mathcal{V}$
f_C	local function defined on the set C
$\varphi_C(x_C)$	factor or potential indexed with a maximal clique $C \in \mathcal{C}$, such that φ_C depends on x only through x_C
Z	$:= \int_{\mathcal{X}} \prod_{C \in \mathcal{C}(\mathcal{G})} \varphi_C(x_C) dx$ partition function, normalizing constant
$\pi(i)$	the set of all parents of a node i
$\mathcal{N}(i)$	the set of all neighbors of a node i
$\mathbb{R}$	the set of real numbers

$\overline{\mathbb{R}}$	$: \mathbb{R} \cup \{\infty\}$ the extended set of real numbers
w	$: \mathcal{E} \to \mathbb{R}$ weighting function from the set of the edges $\mathcal{E}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to the set of real numbers $\mathbb{R}$
$A \cup B$	union of the sets A and B
$A \cap B$	intersection of the sets A and B
$A \setminus B$	difference of the sets A and B, elements in the set A which do not belong to the set B
$A \perp\!\!\!\perp_{\mathcal{G}} B \mid C$	set C is separating the disjoint sets A and B, for which $A, B \subset \mathcal{V} \setminus C$, in the case when the sets are random variables, the relation is for conditional independence of A and B, given C
ϕ_α	$: \mathcal{X} \to \mathbb{R}^d$ function called sufficient statistics
θ_α	canonical parameter for ϕ_α
Θ	space of canonical parameters
$A(\theta)$	$: \mathbb{R}^d \to \overline{\mathbb{R}}$ log-partition function
$A^*(\mu)$	$: \mathbb{R}^d \to \overline{\mathbb{R}}$ conjugate dual to the log-partition function A
$\mathbb{E}[f(x)]$	expected value of a function $f(x)$
$\mathcal{M}$	mean parameter space
μ_α	mean parameter
$\mathcal{M}^o$	interior of (the mean parameter) space
$\text{rint}(\mathcal{M})$	relative interior of (the mean parameter) space
$\overline{\mathcal{M}}$	closure of (the mean parameter) space
H	$: \mathbb{R}^d \to \mathbb{R}$ Shannon entropy
$\mathcal{M}(\mathcal{G})$	marginal polytope corresponding to the graph $\mathcal{G}$
$\mathcal{L}(\mathcal{G})$	local polytope corresponding to the graph $\mathcal{G}$
$\mathcal{T}$	a tree-like graph
$\mathcal{L}$	set of labels
E_θ	energy function corresponding to a distribution with canonical parameters θ
$[n]$	$= \{1, 2, ..., n\}$ set of all natural numbers i such that $1 \leq i \leq n$
$[.]$	Iverson bracket, 1 if the value in the bracket is true and 0 otherwise
$I_A(x)$	$= \begin{cases} 0 & \text{if } x \in A \\ +\infty & \text{if } x \notin A \end{cases}$, indicator set function on a set A
KL	Kullback-Leibler divergence
$\mathbb{N}$	set of natural numbers
$\|\|x\|\|$	$= \sqrt{\langle x, x \rangle}$ Euclidean norm for $x \in \mathbb{R}^n$
$\langle x, y \rangle$	$= \sum_{i=1}^n x_i y_i$ inner product in the Euclidean space, $x, y, \in \mathbb{R}^n$
$\mathbb{S}_+^d$	cone of symmetric positive semi-definite matrices
$\text{conv}(A)$	convex hull of a set $A \subset \mathbb{R}^n$
$\text{aff}(A)$	affine hull of a set $A \subset \mathbb{R}^n$
$\mathbb{B}(x, \varepsilon)$	open Euclidean ball
$\text{epi}(f)$	epigraf of a function f
$\partial f(x)$	subdifferential of a function f

∇f	gradient of a function f
$\nabla^2 f$	Hessian of a function f
$S_-(f, c)$	sublevel set of a function f
$S_+(f, c)$	superlevel set of a function f

Metric Learning

$\mathcal{G}$	a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ which is a pair of a set of nodes (vertices) $\mathcal{V}$, and a set of edges $\mathcal{E}$
$\mathcal{S}$	indices of similar pairs of points
$\mathcal{D}$	indices of dissimilar pairs of points
$\mathcal{R}$	$:= \{(i, j, k) \,\|\, x_i$ is more similar to x_j than to $x_k\}$
M	an arbitrary matrix in $\mathbb{R}^{n \times n}$
$l(M, \mathcal{D}, \mathcal{S}, \mathcal{R})$	a loss function
$r(M)$	a regularizer function
$\mathbb{S}_+^n$	cone of symmetric PSD $n \times n$ real-valued matrices
I	identity matrix
$\mathrm{tr}(M)$	trace of a matrix M
$d_M(x_i, x_j)$	$:= \langle (x_i - x_j), M(x_i - x_j) \rangle^{1/2}$ distance metric with respect to a matrix M
X_{ij}	$:= (x_i - x_j)(x_i - x_j)^T$
$X_{\mathcal{S}}$	$:= \sum_{(x_i, x_j) \in \mathcal{S}} X_{ij}$
$X_{\mathcal{D}}$	$:= \sum_{(x_i, x_j) \in \mathcal{D}} X_{ij}$
$\triangle_{\lvert\mathcal{D}\rvert - 1}$	$:= \{u \in \mathbb{R}^{\lvert\mathcal{D}\rvert} \,\big\vert\, u_i \geq 0, \sum_{i=1}^n u_i = 1\}$ the $(\lvert\mathcal{D}\rvert - 1)$ dimensional probability simplex
$\tilde{M}$	$:= X_{\mathcal{S}}^{1/2} M X_{\mathcal{S}}^{1/2}$
$\tilde{X}_{ij}$	$:= X_{\mathcal{S}}^{-1/2} X_{ij} X_{\mathcal{S}}^{-1/2}$
$\mathcal{P}$	$:= \{M \vert M \succeq 0$ and $\mathrm{tr}(M) \leq 1\}$
$\mathcal{C}$	$:= \{\tilde{M} \succeq 0 : \mathrm{tr}(\tilde{M}) = 1\}$
$\otimes$	Kronecker product
$I : \mathbb{R} \to \mathbb{R}$	an indicator function defined by $I(x) := \begin{cases} 0 & \text{if } x \geq 0 \\ \infty & \text{if } x < 0 \end{cases}$

Model Parameter Perturbation and Learning

$E_{\theta(x)}$	discrete energy function with potentials $\theta(x)$
$\mathcal{G}$	a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ which is a pair of a set of nodes (vertices) $\mathcal{V}$, and a set of edges $\mathcal{E}$
$\mathcal{L}$	the set of labels
μ	vector of assignments
μ^*	vector of assignments corresponding to groundtruth segmentation
$\mathcal{L}_{\mathcal{G}}$	local polytope defined on a graph $\mathcal{G}$

$\mathcal{L}_{\mathcal{G}}^{M}$	local polytope defined on a graph $\mathcal{G}$ corresponding to minimal representation
$\hat{\theta}$	intial potentials
$\tilde{\theta}$	$:= \hat{\theta} + \theta$ corrected potentials
w	$:= (w_u, w_p)^\top$ parameter vector, where w_u is the part corresponding to the unary potentials and w_p the part corresponding to the pairwise potentials
$\Theta(\mu^*)$	$:= \{\tilde{\theta} \in \mathbb{R}^{m+n} \mid \min_{\mu \in \mathcal{L}_{\mathcal{G}}^{M}} \langle \tilde{\theta}, \mu \rangle = \langle \tilde{\theta}, \mu^* \rangle\}$ set of model parameters that correspond to ground truth assignments
f^i	feature vector corresponding to unary term
f^{ij}	feature vector corresponding to a pairwise term
$k(f^i, f^j)$	$:= \sigma_m^2 \exp\Big(-\frac{1}{2\sigma_f^2} \|f^i - f^j\|^2 \Big)$, with σ_f^2,σ_m^2 parameters
$K(F)$	$:= \{k(f^i, f^j)\}_{i,j \in [N]}$
θ_ϕ	$:= \theta - A^\top \phi$, where A is the matrix corresponding to the local polytope equations and ϕ is a dual variable vector
$L(w)$	$:= \max_{\mu \in \mathcal{L}_{\mathcal{G}}} \{\langle -\tilde{\theta}_w, \mu \rangle + \langle \tilde{\theta}_w, \mu^* \rangle\}$ loss function for the LA method, in the case when there is one training image
$\overline{L}$	upper bound on the loss function L
${g^k}_{k \geq 0}$	sequence of subgradients
${f^k}_{k \geq 0}$	sequence of deflected subgradients
P_S	projection on a set S
α_k	step size in subgradient method
% mis	$100 \frac{\sum_p (I(p) \neq I_{gt}(p))}{\lvert p \rvert}$ percentage of all mislabeled pixels, I is the obtained segmented images, I_{gt} is the ground truth segmentation and $\lvert p \rvert$ is the number of pixels
% mis fg	$100 \frac{Area(F \cap F_{gt})}{Area(F \cup F_{gt})}$ percentage of mislabeled foreground pixels, F_{gt} is the ground truth foreground mask and F is the foreground mask of the obtained segmented image

Table 1.1. - Notation used throughout this thesis

2. Background

This chapter will introduce all necessary tools which are required in the remainder of this thesis.

We organize the chapter as follows: in Sect. 2.1 we start with the fundamental concepts and definitions of graphs and probabilistic graphical models. Next, in Sect. 2.2 we include brief overview on some basic concepts in convex analysis and optimization. In Sect. 2.3 we consider the concept of exponential families which provide a theoretical basis for probabilistic graphical models when interpreted as exponential family models. Considering a graphical model as an exponential family member allows us to use convex analysis when exploring graphical models. We apply these concepts to two of the most important basic problems in computer vision, inference in Sect. 2.4 and learning in Sect. 2.5, while exploiting the theory of exponential models. In Sect. 2.6 we address inverse linear programming which is the most important concept used in this thesis.

2.1. Graphical Models

Graphical models or probabilistic graphical models are tools for modeling probabilistic relations between random variables using a graph-like structure. Graphical models are a widely used tool in computer vision, statistics, machine learning and many other fields in science.

We first introduce the basic concepts and definitions in graph theory, most of which we use throughout this work. Next we continue with probability theory which provides the basis for learning probabilistic graphical models.

2.1.1. Graph Theory Used in Image Processing and Analysis

Let us first define what a graph is, [Die12].

Definition 2.1.1. A ***graph*** is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a set of objects represented as nodes (vertices) $\mathcal{V}$, together with binary subsets of distinct nodes, represented by a set of edges, $\mathcal{E} \subset \{ij \in \mathcal{V} \times \mathcal{V} \,|\, i \neq j\}$.

Each edge in the graph represents some relation between two nodes. The edge ij is referred to as ***directed*** edge if $ij \neq ji$. A graph with undirected edges is referred to as ***undirected*** graph and as ***directed*** graph otherwise.

A ***path*** is a sequence of distinct nodes, in which the sequent nodes are adjacent in the graph $\mathcal{G}$. A path from a node to itself which contains each node and edge not more than once, is called a ***cycle***. A graph is either a ***cyclic*** graph if it containts at least one cycle or an ***acyclic*** graph otherwise. An edge between two nodes of

one cycle, which is not part of the cycle is called a ***chord*** of the cycle. A ***chordal graph*** or ***triangulated graph*** is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for which every cycle with length greater than three has a chord. A directed graph with no directed cycles is called ***directed acyclic graph (DAG)***.

A ***clique*** C in an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a subset of the set of vertices $C \subset \mathcal{V}$, such that there is an edge between each of the vertices in C, that is $\forall i, j \in C, i \neq j : ij \in \mathcal{E}$. A ***maximal clique*** is a clique which can not be extended by adding a further node. We denote the set of maximal cliques of a graph $\mathcal{G}$ with $\mathcal{C}(\mathcal{G})$.

Node i is called a child of a node j, and a node j is a parent of a node i, if there exists a directed edge from j to i. We denote the set of all parents of a node i with $\pi(i)$. ***Neighbors*** of a node i are all nodes which are connected to the node i by an edge, that is, $\mathcal{N}(i) := \{j \in \mathcal{V} \,|\, ij \in \mathcal{E}\}$.

In a ***connected graph*** for each node there exists a path to all other nodes. A ***tree*** is an acyclic undirected connected graph. If an acyclic undirected graph is not connected then the graph is called a ***forest*** or union of trees. A ***spanning tree*** of an undirected graph is a subgraph forming a tree which includes all vertices and a minimum number of edges of $\mathcal{G}$.

Any acyclic directed graph can be transformed into an undirected graph. This transformed undirected graph is called ***moral*** graph.

Definition 2.1.2. An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which all parents of each child are linked, and all directed edges are converted into undirected ones is called a ***moral*** graph.

With assigning a weight function $w : \mathcal{E} \to \mathbb{R}$ to the graph, so that each edge ij gets a weight w_{ij}, the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is called ***weighted*** graph.

A graph $\mathcal{G}$ is said to be ***connected*** if there is a path between every pair of vertices in $\mathcal{G}$. A graph which is not connected is said to be ***disconnected***.

Definition 2.1.3. In an undirected connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $C \subset V$ is said to be a ***cut*** or ***separating*** set of $\mathcal{G}$ if removing it renders the graph disconnected. C separating two disjoint sets $A, B \subset \mathcal{V} \setminus C$, in $\mathcal{G}$ is denoted with $A \perp\!\!\!\perp_{\mathcal{G}} B | C$.

Definition 2.1.4. (A, B, C) is a proper ***decomposition*** of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ into three disjoint subsets $A, B, C \subset \mathcal{V}$, $A \cup B \cup C = \mathcal{V}$ such that C is a clique of $\mathcal{G}$ which separates A and B.

A decomposable graph is one for which there exists a sequence of proper decompositions such that all subsets A and B are cliques in $\mathcal{G}$. An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is decomposable if and only if $\mathcal{G}$ is triangulated, see [CDLS07, Theorem 4.4].

A ***clique tree*** is an acyclic undirected graph whose nodes are formed from the maximal cliques of an undirected graph $\mathcal{G}$, which can be cyclic. For an undirected graph $\mathcal{G}$ which is decomposable there always exists a junction tree which is a clique tree with a special property.

Definition 2.1.5. A ***junction tree*** of an undirected graph $\mathcal{G}$ is a clique tree, so that for any two cliques $\mathcal{C}_a$ and $\mathcal{C}_b$, their intersection $\mathcal{C}_a \cap \mathcal{C}_b$ is contained in every clique on the unique path joining them.

2.1.2. Probabilistic Graphical Models

A probabilistic graphical model is a graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a random variable X vector, composed of variables X_i taking values $x_i \in \mathcal{X}_i$, $\forall i \in \mathcal{V}$. The edge set $\mathcal{E}$ is used to imply conditional independences on X and to factorize the underlying probability distribution $p(x)$ with respect to some measure. We will use the same notation $p(x)$ for the density function and the probability distribution of a random variable X, having a value x, that is we denote $P(X = x) = p(x)$, which is in fact the case for the discrete setting.

For a continuous random variable, the integral of the probabilities of all events have to be one, that is $\int_{\mathcal{X}} p(x)dx = 1$, while for a discrete random variable the probabilities of all events have to sum up to one, that is $\sum_{x \in \mathcal{X}} p(x) = 1$. The ***expected value*** of a function $f(x)$ for a continuous random variable is defined as $\mathbb{E}_p[f(x)] := \int_{x \in \mathcal{X}} p(x)f(x)dx$ and for a discrete random variable as $\mathbb{E}_p[f(x)] := \sum_{x \in \mathcal{X}} p(x)f(x)$. For some subset $A \subset \mathcal{V}$, the ***marginal distribution*** of the corresponding continuous variables is $p(x_A) = \int_{\mathcal{X}_{\mathcal{V}\setminus A}} p(x)dx_{\mathcal{V}\setminus A}$ and in the case of discrete variables, $p(x_A) = \sum_{x_{\mathcal{V}\setminus A} \in \mathcal{X}_{\mathcal{V}\setminus A}} p(x)$. For two disjoint subsets $A, B \subset \mathcal{V}$ the ***conditional distribution*** of a random vector X_A, given the observed random vector X_B, with $p(x_B) > 0$ is given with the ***Bayes' rule*** [Bay63]

$$p(x_A|x_B) = \frac{p(x_A, x_B)}{p(x_B)} = \frac{p(x_B|x_A)p(x_A)}{p(x_B)}. \tag{2.1}$$

Let A, B, C be disjoint subsets of $\mathcal{V}$. The random vectors X_A and X_B are called ***conditionally independent***, given X_C with $p(x_C) > 0$, denoted with $X_A \perp\!\!\!\perp X_B|X_C$ if and only if the joint conditional distribution can be written as the product of their marginal conditional distributions, that is

$$X_A \perp\!\!\!\perp X_B|X_C \Leftrightarrow \forall x_A \in \mathcal{X}_A, x_B \in \mathcal{X}_B, x_C \in \mathcal{X}_C : p(x_A, x_B|x_C) = p(x_A|x_C)p(x_B|x_C). \tag{2.2}$$

The ternary relation $X \perp\!\!\!\perp Y|Z$ satisfies the following properties

$$\textbf{\textit{symmetry}} \quad \text{if } X \perp\!\!\!\perp Y|Z \text{ then } Y \perp\!\!\!\perp X|Z \tag{2.3a}$$

$$\textbf{\textit{decomposition}} \quad \text{if } X \perp\!\!\!\perp (Y, Z)|W \text{ then } X \perp\!\!\!\perp Y|W \tag{2.3b}$$

$$\textbf{\textit{weak union}} \quad \text{if } X \perp\!\!\!\perp (Y, Z)|W \text{ then } X \perp\!\!\!\perp Y|(Z, W) \tag{2.3c}$$

$$\textbf{\textit{contraction}} \quad \text{if } X \perp\!\!\!\perp Y|(Z, W) \text{ and } X \perp\!\!\!\perp W|Z \text{ then } X \perp\!\!\!\perp (Y, Z)|W. \tag{2.3d}$$

If the probability distributions p over the random variables X, Y, Z, W are strictly positive then the following additional property holds too

$$\textbf{\textit{intersection}} \quad \text{if } X \perp\!\!\!\perp Y|(Z, W) \text{ and } X \perp\!\!\!\perp W|(Y, Z) \text{ then } X \perp\!\!\!\perp (Y, W)|Z. \tag{2.4a}$$

The notion of a graphical model can be explained using the conditional independences between random variables. Moreover, a graph can be thought as a map of conditional independences between random variables. We consider the following definition of a probabilistic graphical model, valid for both directed and undirected graphical

models :

Definition 2.1.6. A ***probabilistic graphical model*** is a pair $(X, \mathcal{G})$ of a random vector X and a graph $\mathcal{G}$ with a probability distribution $p(x)$. Every conditional independence statement implied by $\mathcal{G}$ is satisfied by $p(x)$, that is for $A, B, C \subset \mathcal{V}$, $A \perp\!\!\!\perp_{\mathcal{G}} B|C \implies X_A \perp\!\!\!\perp X_B|X_C$. The graph $\mathcal{G}$ implies conditional independences between random variables, called Markov property. Moreover the graph $\mathcal{G}$ implies factorization of the probability distribution with

$$p(x) = \prod_{C \in \mathcal{C}} f_C(x_C) \tag{2.5}$$

where $C \subset \mathcal{V}$ and $\mathcal{C} \subset \mathcal{P}(\mathcal{V})$, where $\mathcal{P}(\mathcal{V})$ is the set of all subsets of $\mathcal{V}$, and f_C are local functions defined on the subsets $C \subset \mathcal{V}$.

In the next subsections we give a more precise definition in the sense of the factorization of the probability function when the graphical model is directed and undirected.

2.1.3. Directed Graphical Models: Bayesian Networks

Directed graphical models also referred to as Bayesian networks, make use of the Bayes' rule to factorize the probability distribution such that each factor represents a causal relation of the model, leading to a directed acyclic graph. This relation in directed graphical models is represented by a directed acyclic graph.

Definition 2.1.7. A ***directed graphical model*** $(X, \mathcal{G})$ is a pair of a random vector X and a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, so that the joint probability distribution $p(x)$ of X can be factorized as product of local conditional distributions

$$p(x) = \prod_{i \in \mathcal{V}} p(x_i|x_{\pi(i)}), \tag{2.6}$$

where with $\pi(i)$ we denoted the set of all parents of a node i.

2.1.4. Undirected Graphical Models: Markov Random Fields (MRF)

When the graph structure in the probabilistic graphical model is undirected then the graphical model is commonly called Markov random field (MRF). We first give the definition of an undirected graphical model with respect to the separation property, see Definition 2.1.3.

Definition 2.1.8. An ***undirected graphical model*** $(X, \mathcal{G})$ is a pair of a random vector X and an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ so that if two disjoint subsets $A, B \subset \mathcal{V} \setminus C$ are not connected in $\mathcal{G}_{\mathcal{V} \setminus C}$, then $X_A \perp\!\!\!\perp X_B|X_C$.

The relations between the edge set $\mathcal{E}$ of a graph $\mathcal{G}$ and the random variables are given by the Markov properties of the random variables defined next.

Definition 2.1.9. ***Markov Properties:*** For an undirected graphical model $(X, \mathcal{G})$ we say that X has the

(**G**) ***global Markov property*** with respect to $\mathcal{G}$, if for any three disjoint subsets $A, B, C \subset \mathcal{V}$ for which A and B are separated by C, $X_A \perp\!\!\!\perp X_B | X_C$ holds. This property is satisfied for any undirected graphical model, due to Definition 2.1.8.

(**L**) ***local Markov property*** with respect to $\mathcal{G}$, if $\forall i \in \mathcal{V}$, $X_i \perp\!\!\!\perp X_{\mathcal{V}\setminus(\{i\}\cup\mathcal{N}(i))} | X_{\mathcal{N}(i)}$ holds, where $\mathcal{N}(i)$ denotes the neighborhood of i.

(**P**) ***pairwise Markov property*** with respect to $\mathcal{G}$, if for all pairs of non-adjacent nodes i and j, $X_i \perp\!\!\!\perp X_j | X_{\mathcal{V}\setminus\{i,j\}}$ holds.

If in addition to (**G**) the probability distribution $p(x)$ is strictly positive, $p(x) > 0$ $\forall x \in \mathcal{X}$, then both (**L**) and (**P**) hold for the undirected model $(X, \mathcal{G})$, that is (**G**) is equivalent to (**L**) and (**P**), see [Lau96, CDLS07].

For undirected graphical models there is also a factorization rule of $p(x)$, which is different than the one for directed graphical models.

Definition 2.1.10. For an undirected graphical model $(X, \mathcal{G})$ the probability distribution $p(x)$ of the random vector X factorizes as

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}(\mathcal{G})} \varphi_C(x_C) \quad \text{and } Z = \int_{\mathcal{X}} \prod_{C \in \mathcal{C}(\mathcal{G})} \varphi_C(x_C) dx \tag{2.7}$$

where φ_C are called ***factors*** or ***potentials***, indexed with the maximal cliques $C \subset \mathcal{C}(\mathcal{G})$ and φ_C depends on x only through x_C. The normalizing constant Z is also refered to as ***partition function*** and ensures that the integral of the probabilities of all events is 1.

It can be shown that if X has the factorization property with respect to an undirected graph $\mathcal{G}$, then X has the global Markov property. The reverse holds in the case of strict positivity of $p(x)$ $\forall x \in \mathcal{X}$. This is stated in the following theorem known as Hammersley-Clifford theorem [HC71].

Theorem 2.1.1. ***Hammersley-Clifford :*** *A random vector X with a strictly positive probability distribution, that is $p(x) > 0$ $\forall x \in \mathcal{X}$ satisfies the pairwise Markov property with respect to an undirected graph $\mathcal{G}$ if and only if it factorizes with respect to $\mathcal{G}$ as in (2.7).*

Proof. See [LS03, Theorem 3.9]. □

In general, the conditional independence properties for undirected graphical models are much simpler than those for directed graphical models. We saw that a directed acyclic graph can be transformed into an undirected one using moralization, see Definition 2.1.2. However, during this transformation some information on conditional independence of variables is lost. On the other hand a subclass of undirected graphs, undirected graphs with chordal graph structure, can be transformed into an equivalent directed one. For details we refer the reader to [KF09].

2.2. Basic Concepts in Convex Analysis and Optimization

In this section we want to introduce basic concepts of convex analysis and optimization which will be used throughout this thesis. For more detailed introduction on convex optimization we refer to [Roc70, BV04].

In practice, solving an optimization problem can be very difficult or not possible at all. However, when the problem is convex, the situation might become better. This is also a cause by the fact that convex optimization problems have been studied a lot in the past and there is a huge literature on how to deal and solve efficiently convex problems. However, the key property of a convex problem is that local optimizers are always global ones.

2.2.1. Convex Sets

We start by defining convex sets.

Definition 2.2.1. A set $A \subseteq \mathbb{R}^n$ is ***convex*** if for all $x, y \in A$ and $0 < \alpha < 1$, the point $\alpha x + (1-\alpha)y \in A$.

The ***intersection*** $\cup_{i\in\mathcal{I}} A_i$ of a family of convex sets $\{A_i\}_{i\in\mathcal{I}}$ for an arbitrary index set $\mathcal{I}$ is also a convex set. The ***Cartesian product*** $A_1 \otimes A_2 \otimes ... \otimes A_n$ of a family of convex sets $\{A_i\}_{i\in\mathcal{I}}$ as defined above is a convex set as well. However, the union of convex sets is not necessarily a convex set.

Definition 2.2.2. A set $K \subset \mathbb{R}^n$ is called a ***cone*** if for all $x \in K$, the ray $\{\lambda x \,|\, \lambda > 0\} \in K$. If K is convex then the cone K is called a ***convex cone***.

The cone of ***symmetric positive semi-definite matrices*** is defined by

$$\mathbb{S}^d_+ := \{X \in \mathbb{R}^{d\times d} \,|\, X = X^\top, X \succeq 0\}. \tag{2.8}$$

Similarly with $\mathbb{S}^d_{++}$ we denote the cone of ***positive definite matrices***

$$\mathbb{S}^d_{++} := \{X \in \mathbb{R}^{d\times d} \,|\, X = X^\top, X \succ 0\}. \tag{2.9}$$

Theorem 2.2.1. ***Farkas-Minkowski-Weyl****: A cone is polyhedral if and only if it is finitely generated.*

Proof. For proof see [Sch98, Corollary 7.1a]. □

A ***polytope*** is the convex hull of its vertices or extreme points.

Definition 2.2.3. The ***convex hull*** of a set $A \subset \mathbb{R}^n$ is defined as

$$\mathrm{conv}(A) := \left\{ \sum_{i=1}^n \alpha_i x_i \,|\, A = \{x_1, x_2, ..., x_n\}, \alpha_i \in \mathbb{R}, \alpha_i \geq 0, \sum_{i=1}^n \alpha_i = 1 \right\}. \tag{2.10}$$

Definition 2.2.4. The ***affine hull*** of a set $A \subset \mathbb{R}^n$ is defined as

$$\mathrm{aff}(A) := \left\{ \sum_{i=1}^n \alpha_i x_i \,|\, A = \{x_1, x_2, ..., x_n\}, \alpha_i \in \mathbb{R}, \sum_{i=1}^n \alpha_i = 1 \right\}. \tag{2.11}$$

Due to more restrictions the convex hull is always a subset of the affine hull.

Definition 2.2.5. A $n-1$ dimensional ***simplex*** Δ_{n-1} is a $n-1$ dimensional polytope which is a convex hull of its n dimensional vertices

$$\Delta_{n-1} := \{u \in \mathbb{R}^n : u_i \geq 0, \sum_{i=1}^{n} u_i = 1\}. \tag{2.12}$$

The simplex has one dimension less than the space in which it is embedded due to the constraint that the sum of all its vertices has to sum to one, and so one of the n vertices can be expressed using the rest $n-1$, which makes it one dimension less than $\mathbb{R}^n$.

Definition 2.2.6. For $\varepsilon > 0$ and $x \in \mathbb{R}^n$ the open ***Euclidean ball*** is defined by

$$\mathbb{B}(\varepsilon, x) := \{y \in \mathbb{R}^n \,|\, ||y - x|| < \varepsilon\}. \tag{2.13}$$

Definition 2.2.7. The ***interior*** of a convex set $A \subset \mathbb{R}^n$ is defined by

$$A^o := \{x \in A \,|\, \exists \varepsilon > 0 : \mathbb{B}(\varepsilon, x) \subset A\}, \tag{2.14}$$

where $\mathbb{B}(\varepsilon, x)$ is the Euclidean ball as defined in Definition 2.2.6.

Definition 2.2.8. The ***relative interior*** of a convex set $A \subset \mathbb{R}^n$ is defined by

$$\text{rint}(A) := \{x \in A \,|\, \exists \varepsilon > 0 \text{ s.t. } \mathbb{B}(\varepsilon, x) \cap \text{aff}(A) \subset A\}, \tag{2.15}$$

with $\mathbb{B}(\varepsilon, x)$ as defined in Definition 2.2.6.

The interior A^o and relative interior $\text{rint}(A)$ of a convex set $A \subset \mathbb{R}^n$ are also convex sets and $\text{rint}(A)$ is always nonempty, i.e. $rint(A) \neq \varnothing$. A convex set $A \subseteq \mathbb{R}^n$ is ***full-dimensional*** if $\text{aff}(A) = \mathbb{R}^n$. If A is full-dimensional then $\text{rint}(A) = A^o$.

An important function we will use is the ***set indicator function*** defined on a set $A \subset \mathbb{R}^n$ by

$$I_A(x) := \begin{cases} 0 & \text{if } x \in A \\ +\infty & \text{if } x \notin A. \end{cases} \tag{2.16}$$

When A is convex then I_A is convex too. Indicator functions of convex sets are of interest in constrained convex optimization. With their help, a constrained optimization problem can be converted into an unconstrained one. As a consequence we can treat both constrained and unconstrained convex problems in a unified way.

2.2.2. Convex Functions

Definition 2.2.9. A function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ is ***convex*** if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \tag{2.17}$$

holds for all $x, y \in \mathbb{R}^n$ and $0 < \alpha < 1$.

The ***domain*** of the function f is defined by

$$\operatorname{dom} f := \{x \in \mathbb{R}^n \mid f(x) < \infty\}. \tag{2.18}$$

Function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ is convex if and only if the ***epigraf*** of f defined by

$$\operatorname{epi}(f) := \{(x, y) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq y\} \tag{2.19}$$

is convex.

Definition 2.2.10. A function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ is ***strongly convex*** on some set $A \subset \mathbb{R}^n$ if there exists $m > 0$ such that

$$\nabla^2 f(x) \succeq mI \tag{2.20}$$

with I the identity matrix.

Definition 2.2.11. A convex function f is called ***proper*** if f is finite for at least one point, that is $\operatorname{dom} f \neq \varnothing$.

Continuity

A function f is ***lower semi-continuous*** if for any sequence $\{x_k\}_k \subset \mathbb{R}^n$ so that $\{x_k\}_k \to x \in \mathbb{R}^n$

$$f(x) \leq \liminf_{k \to \infty} f(x_k). \tag{2.21}$$

Similarly f is ***upper semi-continuous*** if for any sequence $\{x_k\}_k \subset \mathbb{R}^n$ so that $\{x_k\}_k \to x \in \mathbb{R}^n$

$$f(x) \geq \limsup_{k \to \infty} f(x_k). \tag{2.22}$$

If f is both lower semi-continuous and upper semi-continuous then f is ***continuous***.

Definition 2.2.12. A function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ is ***continuous*** relative to some convex set $A \subset \mathbb{R}^n$ if the restriction of f to A denoted by $f|_A$ is a continuous function.

A stronger notion for the relative continuity is given by the following definition.

Definition 2.2.13. A function $f : A \to \mathbb{R}$ is called ***Lipschitz continuous*** relative to $A \subset \mathbb{R}^n$ if there exists $L \geq 0$ such that

$$||f(x) - f(y)|| \leq L||x - y|| \tag{2.23}$$

where $x, y \in A$.

We will see that Lipschitz continuity of the gradient of f is the key ingredient for convergence guarantees of some optimization algorithms, including the gradient descent.

Duality

For a proper function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ its ***conjugate dual*** or ***Legendre-Fenchel conjugate dual*** function $f^* : \mathbb{R}^n \to \overline{\mathbb{R}}$ is defined by

$$f^*(y) := \sup_{x \in \mathbb{R}^n} \{\langle x, y\rangle - f(x)\}. \tag{2.24}$$

The dual is always a convex and lower semi-continuous function even if the function f is not convex.

Definition 2.2.14. A convex differentiable function f, is ***essentially smooth***, if it has a nonempty domain $A = \mathrm{dom}(f)$, f is differentiable on A^o and $\lim_{k\to\infty} \nabla f(x_k) = \infty$ for any sequence $\{x_k\}_k \in A$, $\lim_{k\to\infty} x_k = \mu$, where μ is a boundary point of A.

A convex function f with $\mathrm{dom}(f) = \mathbb{R}^n$ is essentially smooth since its domain $\mathbb{R}^n$ has no boundaries.

Differentiability

A convex function is not necessary differentiable. For this reason the notion of differentiability is generalized and to this end the notion ***subdifferential*** of a proper function $f : A \to \overline{\mathbb{R}}$, $A \subseteq \mathbb{R}^n$ is introduced and defined as

$$\partial f(x) := \{w \in A \,|\, f(y) \geq f(x) + \langle w, y - x\rangle, \forall y \in A\}. \tag{2.25}$$

For a convex differentiable f at x the set of subdifferentials at x consists of a single element which is exactly the gradient ∇f. When f is lower semi-continuous, then the set of subdifferentials of f is always nonempty, see [Nes04, Theorem 3.1.13].

Proposition 2.2.2. *A differentiable function $f : A \to \mathbb{R}$, where $A \subset \mathbb{R}^n$ is open, is **convex** if and only if for all $x, y \in A$*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \tag{2.26}$$

is satisfied. If the inequality is strict whenever $x \neq y$, f is strictly convex.

Proof. For proof see [BV04, Sect. 3.1.3]. □

Proposition 2.2.3. *For a twice differentiable function $f : A \to \mathbb{R}^n$ where $A \subset \mathbb{R}^n$ is convex and open, f is convex on A if and only if the Hessian of f defined by*

$$\nabla^2 f(x) := \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x_1, ..., x_n)\right)_{i,j\in[n]}, \quad x = (x_1, .., x_n)^\top \tag{2.27}$$

is positive semi-definite $\forall x \in A$.

Proof. For proof see [HUL93, Theorem 4.3.1]. □

Sublevel set

Definition 2.2.15. The ***sublevel set*** of a function f is defined as

$$S_-(f,c) = \{x \in \mathrm{dom}(f) \,|\, f(x) \leq c\} \tag{2.28}$$

and the ***superlevel set*** accordingly as

$$S_+(f,c) = \{x \in \mathrm{dom}(f) \,|\, f(x) \geq c\}. \tag{2.29}$$

For an initial point x^0 of an iterative algorithm, it should be always satisfied that the point is in the domain of the function $x^0 \in \mathrm{dom}(f)$ and the sublevel set of the function f should be closed. Sublevel sets are important for analyzing convergence of some algorithms like gradient descent: based on the condition number of the Hessian of the function on the sublevel set, the speed of convergence of gradient descent can be determined.

2.2.3. Gradient Descent Method

Gradient descent is a first order method for optimizing a differentiable convex function. Every next iterate is computed in the direction of the negative gradient and with an appropriately chosen step size. For a convex differentiable function $f : \mathbb{R}^N \to \mathbb{R}$, let x^i be the current iterate, then the next iterate is given by

$$x^{i+1} = x^i - \lambda^i \nabla f(x^i) \tag{2.30}$$

where λ^i is a step size appropriately chosen in every iteration. For a descent method we demand that the function value is decreased in every step, i.e.

$$f(x^{i+1}) < f(x^i) \tag{2.31}$$

until the optimal x^* is reached. Due to convexity of f, from Proposition 2.2.2 with $y = x^{i+1}$ and $x = x^i$ and using the previous inequality (2.31) we require for the variable update:

$$\nabla f(x^i)^\top (x^{i+1} - x^i) < 0. \tag{2.32}$$

The convergence of the gradient descent depends on the choice of the step size λ^i. Convergence can be proved when ∇f is Lipschitz continuous with Lipschitz constant $L \geq 0$, if $0 < \inf_i \lambda^i \leq \sup_i \lambda^i < \frac{2}{L}$, see [Nes04].

Line Search Methods

Line search methods search along the line direction for the next iterate, thus determining a step size. In theory, they aim at finding a global minimum of the function

$$g(\lambda^i) = x^i - \lambda^i \nabla f(x^i) \tag{2.33}$$

where $\lambda^i > 0$. Computing this global minimum is in general very expensive and requires computation of the gradient of g in each step. Due to this, line search methods are more efficient when they search only for an approximation to this global minimum or just a sufficient reduction of f in the next iterate.

Line search methods work such that some condition which implies decrease of the function in the next iterate is satisfied. We present the most commonly used conditions in line search methods.

An inexact line search method is said to satisfy the ***Wolfe conditions*** if the following constraints are fulfilled:

$$\textbf{\textit{Armijo condition}} : f(x^i + \lambda^i \nabla f(x^i)) \leq f(x^i) + c_1 \nabla f(x^i)^\top \nabla f(x^i) \tag{2.34a}$$

$$\textbf{\textit{curvature condition}} : \nabla f(x^i + \lambda^i \nabla f(x^i))^\top \nabla f(x^i) \geq c_2 \nabla f(x^i)^\top \nabla f(x^i) \tag{2.34b}$$

where $0 < c_1, c_2 < 1$. The Wolfe conditions are a set of conditions which guarantee sufficiently fast convergence of the gradient descent method. The Armijo condition guarantees sufficient decrease of the function, while the curvature condition excludes short step size which slows down the optimization process. Stronger conditions than the Wolfe conditions are the ***strong Wolfe conditions*** which include the Armijo condition (2.34a), and instead of the curvature condition (2.34b) the ***strong Wolfe condition*** given by

$$|\nabla f(x^i + \lambda^i \nabla f(x^i))^\top \nabla f(x^i)| \geq c_2 |\nabla f(x^i)^\top \nabla f(x^i)| \tag{2.35}$$

with $0 < c_2 < 1$. The strong Wolfe condition assures the iterate to be in the neighborhood of the global optimum of (2.33).

For a function which is differentiable and bounded from below there always exists a step length λ^i so that the Wolfe conditions are satisfied. This is shown in the next Proposition.

Proposition 2.2.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and let $\nabla f(x^i)$ be the descent direction at x^i. Let us also assume that f is bounded from below along the ray $\{x^i + \lambda^i \nabla f(x^i) | \lambda^i > 0\}$. Then for $0 < c_1, c_2 < 1$ there exist intervals of step lengths satisfying the Wolfe conditions from (2.34) as well as the strong Wolfe conditions (2.34a) and (2.35).*

Proof. For proof see [NW06, Lemma 3.1]. □

Commonly used inexact line search method, ***backtracking line search*** works by decreasing the step size λ^i from unit length to $\lambda^i = \beta \lambda^i$, where $0 < \beta < 1$, until the Armijo condition (2.34a) with $0 < c_1 < 0.5$ is satisfied. Backtracking line search can be used for choosing λ^i for the gradient descent method. It is commonly used with the damped Newton method (we will shortly briefly describe it) when there is no guarantee for global convergence.

2.2.4. Subgradient Method

The subgradient method is an optimization algorithm developed first by Shor [Sho85] to minimize non differentiable convex functions. We refer to [Ber99] and [Boy14] for more on subgradient methods.

For the subgradient method a different approach is chosen for selecting the step size. The subgradient direction is not a descent direction, and so the function is not guaranteed to decrease with the next iterate. For a convex function $f : \mathbb{R}^N \to \overline{\mathbb{R}}$, let x^i be the current iterate, then the next iterate with subgradient method is given by

$$x^{i+1} = x^i - \lambda^i g(x^i) \tag{2.36}$$

where λ^i is the step size at the $i-$th iterate and g is some subgradient of f at x^i, that is $g(x^i) \in \partial f(x^i)$ as defined in (2.25). The subgradient direction at an iterate i is given by the negative subgradient $g(x^i)$. In the case of a differentiable function f the subgradient direction is the gradient descent direction.

Even though the subgradient method is not a descent method, an important property that makes the subgradient work is that for certain step size choices the distance from the current iterate to the optimal one is reduced in every step. This is a result from the following Proposition:

Proposition 2.2.5. *Let the iterate x^i be not the optimal one. Then for every optimal solution iterate x^*, we have*

$$||x^{i+1} - x^*|| < ||x^i - x^*||, \tag{2.37}$$

for all step sizes λ^i such that

$$0 < \lambda^i < \frac{2(f(x^i) - f(x^*))}{||g(x^i)||^2}. \tag{2.38}$$

Proof. For proof see [Ber99, Proposition 6.3.1]. □

From the proposition above the range of appropriate step sizes can be known when the optimal value $f(x^*)$ is known. However in the case when this value is not known, an approximation can be used. Using the result from the Proposition above and an approximate estimation of the optimal value $f(x^*)$ allows one to use certain step size choices. For more on how the most common step size choices, were developed we refer to [Ber99, Sect. 6.3.1]. Among the most common step sizes used are: ***constant step size*** when

$$\lambda^i = \text{const} \quad \forall i, \tag{2.39}$$

constant step length when

$$\lambda^i = \frac{\text{const}}{||g(x^i)||_2}, \tag{2.40}$$

square summable but not summable when

$$\sum_{i=1}^{\infty}(\lambda^i)^2 < \infty \quad \text{and} \quad \sum_{i=1}^{\infty}\lambda^i = \infty, \tag{2.41}$$

non-summable diminishing when

$$\lim_{i\to\infty}\lambda^i = 0 \quad \text{and} \quad \sum_{i=1}^{\infty}\lambda^i = \infty. \tag{2.42}$$

For all these step sizes convergence or approximate convergence can be shown, see [Boy14]. For the constant step size and constant step length the subgradient algorithm finds an $\varepsilon-$suboptimal point after a finite number of iterations, that is if $\overline{f}^i$ is the minimal value after the $i-$th iterate of the algorithm and f^* is the exact optimal value then it holds

$$\lim_{i\to\infty}\overline{f}^i - f^* < \varepsilon. \tag{2.43}$$

While for the other step sizes listed, the square summable but not summable and the non-summable diminishing it holds

$$\lim_{i\to\infty} f(x^i) = f^*. \tag{2.44}$$

2.2.5. Newton Method

Newton method is an iterative optimization method for a twice differentiable strictly convex function $f : \mathbb{R}^n \to \mathbb{R}$ with a search direction or ***Newton step*** given with

$$\Delta x = -\nabla^2 f(x)^{-1}\nabla f(x). \tag{2.45}$$

It is a descent method since

$$\nabla f(x^i)^\top \Delta x < 0, \tag{2.46}$$

except when x is optimal, $\nabla f(x) = 0$. The descent direction follows from (2.45) and the positive definiteness of the Hessian matrix of a strictly convex function.

Newton method was developed using the idea of second order Taylor approximation of a function $f : \mathbb{R}^n \to \mathbb{R}$ at x given with

$$\hat{f}(x+y) = f(x) + \nabla f(x)^\top y + \frac{1}{2}y^\top \nabla^2 f(x) y. \tag{2.47}$$

Computing the gradient of the convex function above with respect to y, gives the minimizer $y = \Delta x$, (2.45). Then $\hat{f}(x + \Delta x)$ is an approximation of the minimizer of f.

The ***pure Newton method*** uses a constant and fixed step size $\lambda^i = 1, \forall i$. While the so called ***damped Newton method*** uses a step size using line search methods.

Necessary and sufficient condition for convergence of the Newton method for a twice differentiable function f is that the Hessian of f is Lipschitz continuous with a constant L, f is strongly convex which is equivalent to $\nabla^2 f(x) \succeq mI$, for $m > 0$,

$x \in S_-(f(x), f(x^0))$ and $\nabla^2 f(x) \preceq MI$, $M > 0$, $x \in S_-(f(x), f(x^0))$. The required constants M, m and L are not always known in practice. However convergence analysis for Newton method can be done easily if the function to be optimized is a self-concordant function.

Definition 2.2.16. ***Self-concordant*** function $f : \mathbb{R} \to \mathbb{R}$ is a convex function for which

$$|f'''(x)| \leq f''(x)^{3/2}. \tag{2.48}$$

$f : \mathbb{R}^n \to \mathbb{R}$ is self-concordant if it is self concordant on every line in $\operatorname{dom} f$, that is if $\overline{f}(y) = f(x + yv)$ is self concordant $\forall x \in \operatorname{dom} f$ and $\forall v \in \mathbb{R}^n$.

For self-concordant functions the constant unit step size can be used since then the Armijo condition (2.34a) always holds. As a consequence no computational expensive line search is required.

2.3. Exponential Families

In previous literature it was shown that a specific subclass of probabilistic graphical models can be interpreted as exponential families, see [WJ08]. In this work we will concentrate on discrete probabilistic graphical models by studying them from the exponential family point of view which allows to apply methods and results from convex analysis. That is, we aim at solving a problem represented as a graphical model by transferring it to a convex optimization problem.

In this section we present the most important definitions and results which we consider important and which are used in this thesis. For further details and background we refer to [WJ08] on which also most of the theoretical results in this section are based on. Furthermore, we introduce important properties of exponential models. We remark that we use the same notation from the first section, $p(x)$ is the same as $P(X = x)$.

2.3.1. Basic Definitions and Notions of Exponential Families

Definition 2.3.1. Let X be a random vector with values $x \in \mathcal{X}$ and let $\phi = (\phi_\alpha)_{\alpha \in \mathcal{I}}$ be the associated collection of functions called sufficient statistics, $\phi_\alpha : \mathcal{X} \to \mathbb{R}^d$, where d is the dimension of the index set $\mathcal{I}$, $d = |\mathcal{I}|$. Let $\theta = (\theta_\alpha)_{\alpha \in \mathcal{I}}$, be the corresponding exponential parameter for ϕ, which parametrizes the distribution according to the sufficient statistic ϕ. A distribution with density

$$p_\theta(x) = \exp(\langle \theta, \phi(x) \rangle - A(\theta)), \tag{2.49}$$

taken with some base measure ν is called an ***exponential family model*** generated by the sufficient statistic ϕ, with parameter space

$$\Theta = \{\theta \in \mathbb{R}^d : \int_{x \in \mathcal{X}} \exp(\langle \theta, \phi(x) \rangle) \nu(dx) < \infty\}, \tag{2.50}$$

where

$$A(\theta) = \log \int_{x \in \mathcal{X}} \exp(\langle \theta, \phi(x) \rangle) \nu(dx) \tag{2.51}$$

is called the ***log-partition function*** or ***cumulant function*** and is used for normalization, that is when $A(\theta) < \infty$, then $\int_{\mathcal{X}} p_\theta(x) \nu(dx) = 1$.

For a given fixed sufficient statistic, each member θ from the parameter space (2.50) represents one exponential family model.

If the parameter space (2.50) is open, the exponential family model (2.49) is called ***regular***. We remark that in the rest only regular exponential families will be considered and so we always assume that Θ is open. The exponential family model (2.49) is called ***minimal*** if for the vector of sufficient statistics ϕ there exists no nonzero vector $a \in \mathbb{R}^d$, so that $\langle a, \phi(x) \rangle$ is constant almost everywhere ($\nu-$almost everywhere). In the case of a minimal representation each exponential model is represented by a unique parameter θ. If the exponential family model (2.49) is not minimal, then it is ***overcomplete***. In the case of an overcomplete representation each distribution can be represented by an affine subset of parameter vectors θ.

Two examples for graphical models which are as well exponential families are the Ising model known from statistical physics [Isi25] and Gaussian Markov random fields [SK86].

Example 2.3.1. [Ising Model] The Ising model is a graphical model with graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and state space $\{-1, 1\}^{|\mathcal{V}|}$ represented as the exponential family model

$$p_\theta(x) = \exp\Big(\sum_{i \in \mathcal{V}} \theta^i x_i + \sum_{ij \in \mathcal{E}} \theta^{ij} x_i x_j - A(\theta) \Big) \tag{2.52}$$

where θ^{ij} is constant for all edges for the standard Ising model and the log-partition function is given by

$$A(\theta) = \log \sum_{x \in \{-1,1\}} \exp\Big(\sum_{i \in \mathcal{V}} \theta^i x_i + \sum_{ij \in \mathcal{E}} \theta^{ij} x_i x_j \Big). \tag{2.53}$$

The index set $\mathcal{I}$ for the Ising model is $\mathcal{I} = \mathcal{V} \cup \mathcal{E}$ and so $d = |\mathcal{I}| = |\mathcal{V}| + |\mathcal{E}|$. Due to $A(\theta) < \infty$ for all $\theta \in \Theta$, the Ising model is a regular exponential family model. It is also a minimal exponential model since there are no linear trivial combinations of the parameters θ equal to a constant with respect to a ν counting measure given with $\nu(\{x\}) = 1$ if $x \in \{-1, 1\}$ and $\nu(\{x\}) = 0$ otherwise. The Ising model is used a lot in image analysis, with the binary state space corresponding to 2 values in the image, for image segmentation for instance. In the general case the pairwise interactions can be extended to higher order interaction.

Definition 2.3.2. If θ and θ' define the same distribution, that is $p_\theta(x) = p_{\theta'}(x)$, then θ' is called a ***reparameterization*** of θ.

The vector parameters θ from (2.50) are called ***canonical parameters***. A probability distribution in general can be parametrized using other type of parameters, called ***mean parameters***.

Definition 2.3.3. ***Mean parameters*** $\mu = (\mu_\alpha)_{\alpha\in\mathcal{I}}$ associated with a sufficient statistic $\phi = (\phi_\alpha)_{\alpha\in\mathcal{I}}$ are defined by

$$\mu := \mathbb{E}_p[\phi(X)] = \int_{\mathcal{X}} \phi(x)p(x)\nu(dx), \tag{2.54}$$

where p is an arbitrary density function, not necessary the exponential family model.

Definition 2.3.4. The ***mean parameter space*** is defined by

$$\mathcal{M} := \{\mu \in \mathbb{R}^d \,|\, \exists p(x) \text{ s.t. } p(x) \geq 0, \int_{\mathcal{X}} p(x)\nu(dx) = 1, \mu = \mathbb{E}_p[\phi(X)]\} \tag{2.55}$$

where again $p(x)$ is not necessarily an exponential model.

There is a connection between the mean and canonical parameters through the log-partition function. For this reason we first explore the log-partition function by defining some of its properties.

Let us first introduce some definitions necessary for the theorem to follow.

Definition 2.3.5. For a random variable X with real values, its ***variance*** $\mathrm{Var}(X)$ is defined by

$$\mathrm{Var}(X) := \mathbb{E}((X - \mathbb{E}(X))^2) = \mathbb{E}(X^2) - \mathbb{E}(X)^2. \tag{2.56a}$$

(2.56b)

Definition 2.3.6. For a random variable or a vector X, its ***covariance*** matrix is defined by

$$\mathrm{Cov}(X) := \mathbb{E}[(X - \mathbb{E}(X))(X - \mathbb{E}(X))^\top] = \mathbb{E}(XX^\top) - \mathbb{E}(X)\mathbb{E}(X)^\top. \tag{2.57}$$

Theorem 2.3.1. *The log-partition function*

$$A(\theta) = \log \int_{x\in\mathcal{X}} \exp(\langle\theta, \phi(x)\rangle)\nu(dx) \tag{2.58}$$

associated with any regular exponential family has the following properties:

(a) It has derivatives of all orders on its domain Θ defined in (2.50). The first two derivatives yield the first two moments of the random vector $\phi(X)$, mean and variance:

$$\frac{\partial A}{\partial\theta_\alpha}(\theta) = \mathbb{E}_{p_\theta}[\phi_\alpha(X)] \tag{2.59a}$$

$$\frac{\partial^2 A}{\partial\theta_\alpha\partial\theta_\beta}(\theta) = \mathbb{E}_{p_\theta}[\phi_\alpha(X)\phi_\beta(X)] - \mathbb{E}_{p_\theta}[\phi_\alpha(X)]\mathbb{E}_{p_\theta}[\phi_\beta(X)] \tag{2.59b}$$

(b) $A(\theta)$ is a convex function of θ on its domain Θ. Moreover $A(\theta)$ is strictly convex if the exponential family is minimal.

Proof. For proof see [WJ08, Proposition 3.1.]. □

Crucial result of the Theorem 2.3.1 is that $\nabla A(\theta) = \mathbb{E}_{p_\theta}[\phi_\alpha(X)] = \mu_\alpha$, that is ∇A is a mapping from the space of canonical parameters Θ to the space of mean parameters $\mathcal{M}$. The derivative of the conjugate dual of A on the other hand defines a mapping from the space of mean parameters to the space of canonical parameters. Before exploring these two mappings we first state some properties of the space of mean parameters $\mathcal{M}$.

2.3.2. Properties of the Space of Mean Parameters $\mathcal{M}$

The space $\mathcal{M}$ as defined in (2.55) is always a convex subset of $\mathbb{R}^d$. For two different $\mu_1, \mu_2 \in \mathcal{M}$, there exist distributions p_1 and p_2 that realize them, that is $\mu_1 = \mathbb{E}_{p_{1\theta}}[\phi(X)]$ and $\mu_2 = \mathbb{E}_{p_{2\theta}}[\phi(X)]$. For $0 \leq \lambda \leq 1$, the convex combination $\mu_1\lambda + \mu_2(1-\lambda) \in \mathcal{M}$ and it is realized by the distribution $p_1\lambda + p_2(1-\lambda)$. The properties of the space $\mathcal{M}$ apart from its convexity are given with the following proposition:

Proposition 2.3.2. *For the set of mean parameters $\mathcal{M}$ it holds:*
(a) $\mathcal{M}$ is full-dimensional if and only if the exponential family is minimal.
(b) $\mathcal{M}$ is bounded if and only if for the space of canonical parameters $\Theta = \mathbb{R}^d$ is satisfied and the log-partition function A is globally Lipschitz on $\mathbb{R}^d$.

Proof. See [WJ08, Proposition B.1]. □

2.3.3. Properties of the Forward Mapping ∇A

We now want to see what kind of mapping is the mapping from the space of canonical to the space of mean parameters, $\nabla A : \Theta \to \mathcal{M}$. More precisely, we want to know when or under which conditions the mapping ∇A is bijective.

Proposition 2.3.3. *The mapping $\nabla A : \Theta \to \mathcal{M}$ from the space of canonical parameters to the space of mean parameters is injective if and only if the exponential family defined by the parameters in Θ is minimal.*

Proof. For proof see [WJ08, Proposition 3.2.].

□

Proposition 2.3.4. *For a minimal exponential family the mapping ∇A is surjective onto $\mathcal{M}^o$, the interior of the space of mean parameters $\mathcal{M}$.*

Proof. For proof see [WJ08, Proposition B.1]. □

From these two propositions it follows that for a minimal exponential family, the mapping ∇A is bijective on $\mathcal{M}^o$. That is, a mean parameter, $\mu \in \mathcal{M}^o$ is uniquely determined by the minimal exponential model p_θ. This result can be extended for the case of an overcomplete representation. Let ϕ be a vector of sufficient statistics in the overcomplete representation. After eliminating elements of ϕ until no affine dependencies remain, ϕ can be brought into a vector of sufficient statistics, ψ in an equivalent minimal representation. Let ∇A_ϕ and ∇A_ψ be the corresponding mean parameter mappings and $\mathcal{M}_\phi$ and $\mathcal{M}_\psi$ the corresponding mean parameter spaces.

Then as a result from Proposition 2.3.4 we have that ∇A_ψ is surjective on the interior of $\mathcal{M}_\psi$, $\mathcal{M}_\psi^o$. On the other hand each member from the relative interior of $\mathcal{M}_\phi$, $\text{rint}(\mathcal{M}_\phi)$ see Defintion 2.2.8 for a definition of a relative interior, is related to a unique element from the interior of $\mathcal{M}_\psi$, $\mathcal{M}_\psi^o$. As a conclusion for an overcomplete vector of sufficient statistics ϕ, the corresponding mapping ∇A_ϕ is surjective on the relative interior of $\mathcal{M}_\phi$, $\text{rint}(\mathcal{M}_\phi)$.

Although while defining $\mathcal{M}$ we stated that for the mean parameters the distribution realizing them does not have to be an exponential model from Propositions 2.3.3 and 2.3.4, it follows that for all $\mu \in \mathcal{M}^o$, where $\mathcal{M}^o$ denotes the interior of $\mathcal{M}$, can be realized by an exponential family model. What is distinguishing about the exponential family, and we will see in the next sections, is that it has the maximum entropy of all distributions that realize μ. This is indeed the motivation of using exponential families when defining $\mathcal{M}$.

Concerning entropy and maximum entropy we refer to the following two definitions.

Definition 2.3.7. ***Shannon entropy*** or just entropy is defined as a function of the distribution p

$$H(p) := -\int_{\mathcal{X}} (\log p(x)) p(x) \nu(dx). \tag{2.60}$$

Definition 2.3.8. ***Principle of maximum entropy*** is to choose among distributions which are consistent with the data, the distribution p^* which has the maximum Shannon entropy. That is

$$p^* := \arg\max_{p \in \mathcal{P}} H(p) \text{ subject to } \mathbb{E}_p[\phi_\alpha(X)] = \mu_\alpha \text{ for all } \alpha \in \mathcal{I} \tag{2.61}$$

where $\mathcal{P}$ is the set of all distributions over a random variable X and $\mu = (\mu_\alpha)_{\alpha \in \mathcal{I}}$ is a vector of empirical expectations.

2.3.4. Properties of the Inverse Mapping ∇A^*

Let us first define A^*, the conjugate dual of A. The ***conjugate dual function*** A^* of A is defined by

$$A^*(\mu) := \sup_{\theta \in \Theta} \{\langle \mu, \theta \rangle - A(\theta)\}, \tag{2.62}$$

where μ is a vector of dual variables of the same dimension as $\theta \in \mathbb{R}^d$. For general A, the conjugate dual A^* can take values in $\mathbb{R} \cup \{\infty\}$. However, if A is chosen as a log-partition-function, then it turns out that μ is in fact the mean parameter of $\mathcal{M}$. Furthermore, we will notice that the inverse mapping from the space of mean parameters $\mathcal{M}$ to the space of canonical parameters Θ is in fact ∇A^* as summarized in the following Proposition.

Proposition 2.3.5. *The dual function A^* is convex and lower semi-continuous. In the case when A is a log-partition function of a regular and minimal exponential family the following properties of A^* hold:*

(a) A^* is differentiable on the interior of $\mathcal{M}$, $\mathcal{M}^o$ and $\nabla A^*(\mu) = (\nabla A)^{-1}(\mu)$.

(b) A^* is strictly convex and essentially smooth, see 2.2.14 for a definition of essentially smooth function.

Proof. For proof see [WJ08, Proposition B.2]. □

From the definition of the Shannon entropy in 2.3.7 and the definition of the expectation we have

$$-H(p_{\theta(\mu)}) = \mathbb{E}_{p_{\theta(\mu)}}[\langle\theta(\mu), \phi(X)\rangle - A(\theta(\mu))] = \langle\theta(\mu), \mu\rangle - A(\theta(\mu)), \tag{2.63}$$

where we use the linearity of the expectation and $\mathbb{E}_{p_{\theta(\mu)}}[\phi(X)] = \mu$. From the surjectivity of ∇A, $(\nabla A)^{-1}$ is not empty. In the definition of the dual conjugative function, see (2.62), the supremum is attained for the optimal value $\theta^* \in (\nabla A)^{-1}(\mu)$, $\mu \in \mathcal{M}^o$, such that in the case of minimal representation θ^* is unique, while in the case of an overcomplete representation the optimal θ^* is an affine subset. And so the optimal value (2.62) is given by

$$A^*(\mu) = \langle\theta^*, \mu\rangle - A(\theta^*), \quad \mu \in \mathcal{M}^o. \tag{2.64}$$

From this result together with (2.63) it becomes clear that for $\mu \in \mathcal{M}^o$ the conjugate dual A^* is in fact the negative Shannon entropy. We saw that the result can be extended to overcomplete representations, i.e. in the general case of a regular exponential family when $\mu \in \text{rint}(\mathcal{M})$, A^* is the maximum negative entropy. This is stated in part of the following theorem.

Theorem 2.3.6. *Let $\mu \in \mathcal{M}^o$ and let $\theta(\mu)$ be the unique canonical parameter fulfilling $\nabla A(\theta(\mu)) = \mu$. Then the conjugate dual A^* has the form*

$$A^*(\mu) = \begin{cases} -H(p_{\theta(\mu)}) & \text{if } \mu \in \mathcal{M}^o \\ \infty & \text{if } \mu \notin \overline{\mathcal{M}} \end{cases} \tag{2.65}$$

while for $\mu \in \overline{M} \setminus \mathcal{M}^o$, $A^*(\mu) = \lim_{k\to\infty} A^*(\mu^k)$ where $\{\mu^k\} \subset \mathcal{M}^o$ and $\lim_{k\to\infty} \mu^k = \mu$.

Proof. For a proof see [WJ08, B.2]. □

The log-partition function can be rewritten in terms of the conjugate dual A^*. Using the fact that it is convex, proper and continuous we obtain the following form:

$$A(\theta) = \sup_{\mu\in\mathcal{M}} \{\langle\theta, \mu\rangle - A^*(\mu)\}. \tag{2.66}$$

Theorem 2.3.7. *For all canonical parameters $\theta \in \Theta$ the supremum of (2.66) is attained at a unique vector $\mu \in \mathcal{M}^o$ specified by*

$$\mu = \int_{\mathcal{X}} \phi(x) p_\theta(x) \nu(dx) = \mathbb{E}[\phi(X)]. \tag{2.67}$$

Proof. For a proof see [WJ08, B.2]. □

From Theorem 2.3.6 we can see the dual relationship of A^* and the Shannon entropy, H_{p_θ}. For the case $A^*(\mu) = \infty$ the maximum entropy problem happens to be infeasible. If we would restrict the domain of A^* to $\mathcal{M}$, then we are guaranteed to find θ, or exponential family model that realizes μ and has the maximum entropy.

From the previous results on A^* and properties of ∇A for a minimal representation, in view of Theorem 2.3.6 and 2.3.7, ∇A^* is bijective from $\mathcal{M}^o$ to Θ and similarly for an overcomplete representation, ∇A^* is bijective from $\text{rint}(\mathcal{M})$ to Θ. For further discussion and results on this topic we refer to [WJ08].

2.3.5. Exponential Families for Discrete Graphical Models

When the set $\mathcal{X}$ is finite the space of mean parameters $\mathcal{M}$ can be represented as a convex hull (see Definition 2.2.3) of vector of sufficient statistics, i.e.

$$\mathcal{M} = \text{conv}\{\phi(x), x \in \mathcal{X}\}. \tag{2.68}$$

Due to this representation, the set $\mathcal{M}$ is also called convex polytope, or marginal polytope. Every vertex of the marginal polytope $\mathcal{M}$ corresponds to a value in the finite set $\mathcal{X}$. An equivalent representation of a convex polytope is using an intersection of half-spaces

$$\mathcal{M} = \{\mu \in \mathbb{R}^d | \langle a_j, \mu \rangle \geq b_j, \forall j \in \mathcal{J}\}, \tag{2.69}$$

see Theorem 2.2.1. The non-redundant inequality constraints in this representation are the facets of the marginal polytope.

The forward mapping from the space of canonical to the space of mean parameters in the discrete setting can be interpreted as inference, which we discuss in Sect. 2.4. In contrast, the inverse mapping from the space of mean parameters to the space of canonical parameters can be interpreted as learning, often solved using maximum likelihood estimation (MLE), by maximizing the logarithm of the density $p_\theta(x)$, i.e.

$$\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i) = \sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \langle \theta, \phi(x_i) \rangle - A(\theta) = \tag{2.70a}$$

$$= \sup_{\theta \in \Theta} \langle \theta, \hat{\mu} \rangle - A(\theta) = A^*(\hat{\mu}) = -H_{p_\theta(\hat{\mu})} \tag{2.70b}$$

where $\hat{\mu} := \hat{\mathbb{E}}[\phi(X)] = \frac{1}{n} \sum_{i=1}^{n} x_i$. We saw that in the case of a minimal exponential family and $\hat{\mu} \in \mathcal{M}^o$ the maximum likelihood is unique. Finding this unique optimal solution is equivalent to computing the inverse mapping from the set $\mathcal{M}$ to Θ and for $\hat{\mu} \in \mathcal{M}^o$ and minimal exponential family it is the dual to optimizing the maximum entropy solution. That is, for an exponential family we get the canonical parameters $\theta \in \Theta$ that realize $\mu \in \mathcal{M}^o$ so that we have an MLE estimate and maximum entropy. Computing the MLE estimate in the general case is computationally intensive since it involves computing the log-partition function A. With the results of this Sect. 2.3 we saw that the effort for computing the log-partition function depends on the complexity of the space of mean parameters, $\mathcal{M}$. In fact the complexity of $\mathcal{M}$ grows rapidly with increasing graph size, rendering high-dimensional models impossible to

be characterized. Furthermore, the determining of the negative entropy requires the computation of the inverse mapping $(\nabla A)^{-1}$ which again requires high-dimensional integration in most graphical models. Due to the intractability of these steps in most practical cases approximations for computing M and ∇A have been proposed [Shl76, Wer07, WJW05a].

In the remaining work we will restrict ourselves to discrete graphical models and use the discussed interpretation as exponential families. The density $p(x)$ in the case of a discrete undirected graphical model $(X, \mathcal{G})$ is given with $p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}(\mathcal{G})} f_C(x_C)$, as in (2.7). From here the corresponding exponential family is defined by:

$$\mathcal{I}(\mathcal{G}) := \{(C; i) | C \in \mathcal{C}(\mathcal{G}), i \in \mathcal{X}_C\} \tag{2.71a}$$

$$\phi_{(C;i)}(x) := \begin{cases} 1 & \text{if } x_C = i \\ 0 & \text{otherwise} \end{cases} \tag{2.71b}$$

$$\theta(C; i) := \log(f_C(i)) \tag{2.71c}$$

$$A(\theta) := \log(Z) = \log(\sum_{x \in \mathcal{X}} \langle \theta, \phi(x) \rangle). \tag{2.71d}$$

This section introduced the basic notions used in the next sections where we will investigate inference and learning in graphical models which is closely related to MLE.

2.4. Inference

As discussed before, inference on an exponential family model can be interpreted as the forward mapping ∇A. For a discrete probabilistic graphical model $(X, \mathcal{G})$ with a probability distribution $p_\theta(x)$ inference can refer to:

(1) Maximum a posteriori (MAP) inference: compute the most probable configuration $x^* \in X^*$

$$x^* = \arg\max_{x \in \mathcal{X}} p_\theta(x) \tag{2.72}$$

(2) Marginal inference: compute marginal distributions for $A \subset \mathcal{V}$

$$p_\theta(x_A) = \sum_{x_{\mathcal{V} \setminus A} \in \mathcal{X}_{\mathcal{V} \setminus A}} p_\theta(x) \tag{2.73}$$

(3) Maximum a posteriori marginal (MPM) inference: compute most probable configuration for which the random variables of $A \subset \mathcal{V}$ take the value x_A:

$$p^*_\theta(x_A) = \max_{x_{\mathcal{V} \setminus A} \in \mathcal{X}_{\mathcal{V} \setminus A}} p_\theta(x). \tag{2.74}$$

Note that (2) and (3) are closely connected to computing the expectation or mean parameter $\mu = \mathbb{E}_{p_\theta}[\phi(X)]$ defining the exponential model p_θ. For the sufficient statistic $\phi(x)$ as defined in (2.71b), the mean parameters are the corresponding marginal distributions, $\mu_i = \mathbb{E}_{p_\theta}[\phi_{l_i}(X_i)] = P(X_i = l_i; \theta)$ and $\mu_{ij} = \mathbb{E}_{p_\theta}[\phi_{l_i,l_j}(X_i, X_j)] = P(X_i = l_i \wedge X_j = l_j; \theta)$.

Furthermore, (3) can be used to compute (1) which is an integer program (IP) and can be reformulated as

$$\arg\max_{x\in\mathcal{X}}\langle\theta,\phi(x)\rangle, \tag{2.75}$$

since the log-partition function $A(\theta)$ does not depend on x. Moreover the connection between (1) and the log-partition function can be seen from the following theorem. We remark that from now on we denote the marginal polytope from the previous section, (2.68) with $\mathcal{M}(\mathcal{G})$ to denote that it corresponds to the graphical model defined with the graph $\mathcal{G}$.

Theorem 2.4.1. *For all $\theta\in\Theta$, and $\mathcal{M}(\mathcal{G})$ as defined in (2.68), optimization problem (1) has two equivalent representations given with*

$$\max_{x\in\mathcal{X}}\langle\theta,\phi(x)\rangle = \max_{\mu\in\mathcal{M}(\mathcal{G})}\langle\theta,\mu\rangle \tag{2.76a}$$

$$\max_{x\in\mathcal{X}}\langle\theta,\phi(x)\rangle = \lim_{\beta\to\infty}\frac{A(\beta\theta)}{\beta}. \tag{2.76b}$$

Proof. For proof see [WJ08, Theorem 8.1]. □

This theorem shows that the integer program (IP) (2.72), can be transformed to a linear program (2.76a). IPs are NP-hard in general. Due to this, the complexity of the constraint set of the LP, the marginal polytope is apparent. The transformation from IP to LP over the convex hull of its solutions is a standard technique in combinatorial optimization and integer programming, see e.g. [BT97, GLS88]. Solving the MAP inference problem over the marginal polytope is in general computationally intractable due to the complexity of $\mathcal{M}(\mathcal{G})$. In fact the growth of the number of constraints for $\mathcal{M}(\mathcal{G})$ is non-polynomial in the number of nodes of the graph $\mathcal{G}$, see [DL09].

As a simple illustration of the marginal polytope we give an example of the Ising model for a very small graph with 3 nodes and 2 edges.

Example 2.4.1. [Marginal Polytope for the Ising Model] The sufficient statistic for the Ising model are the singleton functions $(x_i, i\in\mathcal{V})$ and the pairwise functions $(x_ix_j, ij\in\mathcal{E})$. The mean parameters correspond to the marginal probabilties, that is for $\mu\in\mathbb{R}^{|\mathcal{V}|+|\mathcal{E}|}$

$$\mu_i = \mathbb{E}_{p_\theta}[X_i] = P(X_i=1) \quad \forall i\in\mathcal{V} \tag{2.77a}$$

$$\mu_{ij} = \mathbb{E}_{p_\theta}[X_iX_j] = P(X_i=1\wedge X_j=1) \quad \forall ij\in\mathcal{E}. \tag{2.77b}$$

The marginal polytope $\mathcal{M}$ is the convex hull of the vector of sufficient statistics. Since $\mathcal{M}$ is the set of all mean parameters it is the set of all singleton and pairwise marginal probabilities realized by some distribution over the random variables $X_i\in\{0,1\}$, $i\in\mathcal{V}$. This marginal polytope for the Ising model is also called correlation or cut polytope. Now let us consider the simple example of a graph with 3 nodes and 2 edges, the variables X_1, X_2, X_3 and the edges X_{12} and X_{23}. The 5 dimensional marginal polytope for the Ising model on this particular graph can be represented by the convex hull of its sufficient statistics, that is the convex hull of

$\{(x_1, x_2, x_3, x_{12}, x_{23}) \,|\, x_i \in \{0, 1\}, i = 1, 2, 3\}$ and thus

$$\mathcal{M}(\mathcal{G}) = \text{conv}\{(0,0,0,0,0), (0,0,1,0,0), (0,1,0,0,0), (1,0,0,0,0), \tag{2.78a}$$
$$(1,1,0,1,0), (1,0,1,0,0,0), (0,1,1,0,1), (1,1,1,1,1)\}. \tag{2.78b}$$

The marginal polytope can also be represented as intersection of half spaces as in (2.69). The half space representation is induced by he constraints on the mean parameters due to (2.77), i.e.

$$\mu_{ij} \geq 0 \tag{2.79a}$$
$$\mu_i - \mu_{ij} \geq 0 \tag{2.79b}$$
$$\mu_j - \mu_{ij} \geq 0 \tag{2.79c}$$
$$\mu_{ij} - \mu_i - \mu_j \geq 1 \tag{2.79d}$$

for $i, j \in \{1, 2, 3\}$, $i < j$, which for our particular graph can be written in the following matrix form:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ -1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_{12} \\ \mu_{23} \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \end{pmatrix}. \tag{2.80}$$

We can see that even for a very simple graph structure the marginal polytope has already many constraints in the half space representation.

NP-hardness of MAP inference in Bayesian networks was proven in [Shi94, Coo90]. However, there exist relaxation methods which efficiently solve the MAP inference problem approximately. The theory of discrete exponential models can be utilized to solve the intractable MAP inference problem by relaxing it to a convex optimization problem.

We remark that usually x is not observed but some noisy observation y, also seen as a random variable, but for compactness of notation we write $p_\theta(x)$ instead of $p_\theta(x|y)$.

In this work we will only consider MAP inference problems. Note that the solution of a MAP inference problem is not unique, in fact there is an affine set of global solutions in the general case of an overcomplete representation as we already saw in the previous sections.

2.4.1. Approximate MAP Inference

The MAP inference problem, (2.72) can be transformed into an equivalent LP (2.76a) which is intractable in the general case due to the complexity of the marginal polytope $\mathcal{M}(\mathcal{G})$. The LP can be relaxed, by replacing the constraint set $\mathcal{M}(\mathcal{G})$ with a convex

superset of it, the so called local polytope, $\mathcal{M}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G})$. Concerning relaxed problems we state the following definition:

Definition 2.4.1. Given two problems

$$(1) \quad \min_{x \in U} f(x) \tag{2.81a}$$

$$(2) \quad \min_{x \in V} g(x) \tag{2.81b}$$

with same variables x, (1) is a relaxation of the problem (2) if and only if $V \subset U$ and $\forall x \in V \ f(x) \leq g(x)$.

Relaxing the integer constraints to real ones and imposing convex constraints on them, the local polytope is defined by

$$\begin{aligned} \mathcal{L}(\mathcal{G}) : &= \begin{cases} \mu \in [0,1]^d \\ \sum_{x_i \in \mathcal{X}_i} \mu_i(x_i) = 1, i \in \mathcal{V} \\ \sum_{x_i \in \mathcal{X}_i} \mu_{ij}(x_i, x_j) = \mu_i(x_i), x_j \in \mathcal{X}_j, ij \in \mathcal{E} \\ \sum_{x_j \in \mathcal{X}_j} \mu_{ij}(x_i, x_j) = \mu_j(x_j), x_i \in \mathcal{X}_i, ij \in \mathcal{E} \end{cases} \\ &= [0,1]^d \cap \text{aff}(\{\phi(x) | x \in \mathcal{X}\}). \end{aligned} \tag{2.82}$$

While for $\mathcal{M}(\mathcal{G})$ the number of constraints grows fast with increasing graph size, the number of constraints for $\mathcal{L}(\mathcal{G})$ is only linear in the graph size. Fig. 2.1 illustrate $\mathcal{L}(\mathcal{G})$ and $\mathcal{M}(\mathcal{G})$ for a simple case.

If the linear programming relaxation has an integer solution, then the solution is exact, and the bound is tight. This is always the case when the graph $\mathcal{G}$ is acyclic. When $\mathcal{G}$ is cyclic, we have $\mathcal{M}(\mathcal{G}) \subset \mathcal{L}(\mathcal{G})$.

One way of solving the relaxed LP over $\mathcal{L}(\mathcal{G})$ is using a software implementation of a general LP solver, which can be inefficient in the case of problems with huge size, since memory consumption grows fast with the problem size. For this reason specialized solvers were developed which utilize the structure of the graphical model. Among the proposed convex relaxations of the MAP problem is the LP relaxation proposed by Schlesinger [Shl76], reviewed in [Wer07], as well as the approach proposed in [SSKS12]. There are also attempts in research to make the relaxation tighter, by posing more constraints in addition to the local polytope $\mathcal{L}(\mathcal{G})$.

Another powerful relaxation technique of solving (2.72) approximately is the so called primal-dual scheme [PKT11]. These algorithms make use of the Lagrangian decomposition known as dual decomposition. The original MAP inference problem is first relaxed to a bi-level optimization problem, where at the lower level the inference is performed on tractable subproblems called slaves. At the higher level the master problem combines the solutions from the slave subproblems via dual variables. The solution to the dual problem of the relaxation of the MAP inference problem is a lower bound to the solution of the primal one, the original MAP problem. Dual decomposition approaches aim at minimizing the gap between the primal and the dual. Depending on the method chosen for decomposing the problem into subproblems

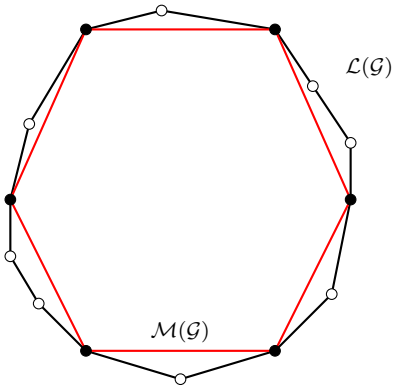


Figure 2.1. - Local, $\mathcal{L}(\mathcal{G})$, black, and marginal polytope $\mathcal{M}(\mathcal{G})$, red. The vertices of the marginal polytope $\mathcal{M}(\mathcal{G})$ correspond to mean parameters. We remark that this is just a qualitative sketch of a local and marginal polytope. The figure can be deceptive in the sense that $\mathcal{L}(\mathcal{G})$ has more facets than $\mathcal{M}(\mathcal{G})$ which is not true. The facets of the marginal polytope are the half-space intersections, the linear inequalities as in (2.69). The additional vertices in $\mathcal{L}(\mathcal{G})$, the non filled vertices are the fractional solutions.

and to solve them, as well as the optimization scheme for the dual problem, different dual decomposition relaxation methods result.

One reason why dual methods have been introduced for MAP inference is that usually the dual (or Lagrange dual) is easier to be solved, since it is always a convex optimization problem, independent on the convexity of the primal.

One possible approach to decompose (2.72) is to choose optimization on trees $\mathcal{T} \subset \mathcal{G}$ as subproblems. Tree reweighted (TRW) algorithms work in this way, they first transform the graph into subtrees, [Kol06, WJW05b]. When the graph is a tree and thus is acyclic, dual decomposition methods are equivalent to message passing and as a consequence we have a tight bound on the relaxation, that is $\mathcal{M}(\mathcal{G}) = \mathcal{L}(\mathcal{G})$.

Other approximate optimization techniques developed to solve the MAP inference problem, are simulated annealing [KGV83], belief propagation [MWJ99] and graph cuts [BVZ01, KR07].

2.4.2. Variational Formulation

In the literature, it is common to consider the minimization of an energy or cost function instead of maximizing a probability. The advantage of using an energy function is the lack of the normalization constant, the log-partition function.

The probability distribution expressed in terms of an energy function, E, is given by:

$$p_\theta(x) = \frac{1}{Z(\theta)} \exp(-E_\theta(x)) \tag{2.83a}$$

where for the normalization constant we have $Z(\theta) = \exp(A(\theta))$, with A being the log-partition function. In the sequel we consider the distribution in terms of the

energy. Due to the equivalence, we can minimize the energy instead of maximizing the probability distribution. Then the MAP inference reads

$$x^* = \arg\min_{x \in \mathcal{X}} E_\theta(x) + \text{const} \tag{2.84}$$

where const $= \log(Z(\theta))$. Note that $\log(Z(\theta))$ is the log-partition function we defined with $A(\theta)$, see (2.51).

2.4.3. Image Labeling Problem

Many important problems from computer vision can be formulated as solving an image labeling problem, for instance image segmentation [AK06, LG12, SM00], image restoration [AK06, LG12], etc. For that reason, in this work we concentrate on the image lableing problem.

The labeling problem is assigning a unique label to each vertex in a graph which along with its neighborhood structure includes the relations among the vertices in the graph. The problem to solve is to find the best labeling, while satisfying conditions incorporated in the graph structure through minimizing a defined energy function on the corresponding graph.

An image is represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes (pixels or superpixels (grouped pixels) and $\mathcal{E}$ is the set of all edges between neighboring nodes which represent some dependencies between nodes. We concentrate on discrete graphical models where each node i receives value $x_i \in \mathcal{X}_i$. We assume that $\mathcal{X}_i$ are the same for all nodes $i \in \mathcal{V}$, which we denote with $\mathcal{L}$. Now we can say that labeling is a mapping $X : \mathcal{V} \to \mathcal{L}$ from the set of nodes, $\mathcal{V}$ to the set of labels $\mathcal{L}$. Translating to an energy function this is a mapping $E : \mathcal{L}^{|\mathcal{V}|} \to \mathbb{R}$, from the set of all possible labellings (events) $\mathcal{L}^{|\mathcal{V}|} = \mathcal{X}$ to the set of real numbers, $\mathbb{R}$.

2.4.4. Graph Cuts

From now on we consider pairwise graphical models, i.e. the size of the cliques is at most two. Furthermore, we restrict ourselves to the case of binary state variables, i.e. the label space has two elements, $\mathcal{L} = \{0, 1\}$. Many problems in computer vision are binary, like image segmentation, binary image restoration and binary image denoising. Furthermore, many energy minimization methods work with reducing the problem into binary subproblems, like the move-making algorithms, the $\alpha - \beta$ swap and the $\alpha-$expansion algorithm [BVZ01].

In the following we consider the energy function on the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in the form

$$E_\theta(x) = \text{const} + \sum_{i \in \mathcal{V}} \theta^i(x_i) + \sum_{ij \in \mathcal{E}} \theta^{ij}(x_{ij}). \tag{2.85}$$

The constant term emerging from the normalization factor is usually neglected in the representation as it does not affect the solution. The function $\theta^i(x_i)$ is called data term and measures the cost of assigning the label x_i to a node $i \in \mathcal{V}$ based on some observed data. The function $\theta^{ij}(x_i, x_j)$ referred to as smoothness term

is used to penalize neighboring nodes having different labels, or in other words it imposes smoothness. It is also a prior term, defined using prior knowledge about the particular problem. The data and smoothness terms are defined application-specific.

Some optimization algorithms for solving the MAP inference problem efficiently, or even with a guarantee of a global optimum require that the energy function satisfies certain conditions. One important property is defined below.

Definition 2.4.2. A binary ***submodular function*** is a function θ, defined on a graph with 2 labels whose pairwise term satisfy

$$\theta^{ij}(1,0) + \theta^{ij}(0,1) \geq \theta^{ij}(0,0) + \theta^{ij}(1,1), \tag{2.86}$$

that is sum of pairwise terms with the same labels is smaller or equal to the sum of the pairwise terms with differing labels. The extension of submodularity to n labels, where the set of labels is given by $\mathcal{L} = \{1, 2, ..., n\}$ is

$$\theta^{ij}(\alpha,\alpha) + \theta^{ij}(\beta,\gamma) \leq \theta^{ij}(\alpha,\gamma) + \theta^{ij}(\beta,\alpha) \tag{2.87}$$

which must hold for all $\alpha, \beta, \gamma \in \mathcal{L}$.

In the binary case it was proven that submodular functions can be optimized globally using graph cuts [KZ04], which turned out to be also the most efficient algorithms for this problem class. The idea is to construct a graph corresponding to the considered energy function such that a solution to the minimum cut problem also solves the energy minimization problem. The minimum cut on the other hand can be efficiently computed with max-flow algorithms, due to the equivalence of these two problems which we will demonstrate shortly. Before stating the main theorem let us first introduce how the terms capacity, flow and cut are defined on the context of a graph.

Capacity is defined as $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$, for $i, j \in \mathcal{V}$ and in particular $c(i,j) = 0$ for $ij \notin \mathcal{E}$.

Let the set of nodes be extended with two additional terminal nodes, s and t, referred to as source and sink, respectively. The ***flow*** on the graph $\mathcal{G}$ is defined as a function $f : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ such that:

a) for all nodes $i, j \in \mathcal{V} \cup \{s,t\}$, $f(i,j) \leq c(i,j)$

b) for all nodes $i, j \in \mathcal{V} \cup \{s,t\}$, $f(i,j) = -f(j,i)$

c) for all nodes $i \in \mathcal{V}$, $\sum_{j \in \mathcal{V} \cup \{s,t\}} f(i,j) = 0$.

The value of the flow in a graph is the sum of the flow leaving the source s, which is equivalent to the sum of the flow entering the sink, t,

$$|f(\mathcal{G})| = \sum_{i \in \mathcal{V} \cup \{s,t\}} f(s,i) = \sum_{i \in \mathcal{V} \cup \{s,t\}} f(i,t). \tag{2.88}$$

The maximum flow is a flow f which maximizes $|f(\mathcal{G})|$.

A ***cut*** in a graph $\mathcal{G}$ is a subset of the edge set, which partitions the node set $\mathcal{V} \cup \{s,t\}$ into two disjoint subsets $\mathcal{V}_s$ and $\mathcal{V}_t$ such that $s \in \mathcal{V}_s$, $t \in \mathcal{V}_t$. The total cost

of a cut is given with

$$\sum_{i \in \mathcal{V}_s, j \in \mathcal{V}_t} c(i,j). \tag{2.89}$$

A minimum cut is one which has minimum costs.

It is common to describe the graph cut and max-flow algorithms using directed graphs. Any undirected graph as considered in this work can be mapped to a directed graph by simply constructing for one undirected two opposing directed edges. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ be a weighted directed graph. A weighted graph can be interpreted as a flow network were the capacity of an edge is given by pairwise weights θ^{ij}. Weight or capacity of an edge is the pairwise term in the energy function formulation, θ^{ij}.

Only for graphs with non-negative weights algorithms with polynomial complexity are known. For the case when some edge weights are negative, a reparameterization as introduced in Definition 2.3.2 allows to bring the energy function into a form having non-negative weights only [KR07]. Furthermore, from Definition 2.3.2 follows that the solution of the minimization problem does not change. When all the weights $\theta^i(x_i) \geq 0$ for $x_i \in \{0,1\}$ and $\min\{\theta^{ij}(0,x_j), \theta^{ij}(1,x_j)\} = 0$ for $x_j \in \{0,1\}$ (which implies that all weights are non-negative) the energy is said to be in ***normal form***.

The normal form is not unique and any binary energy function (2.85) can be brought into normal form in the following way:

1. While there is an edge $ij \in \mathcal{E}$ such that $\min\{\theta^{ij}(0,x_j), \theta^{ij}(1,x_j)\} \neq 0$ and $\epsilon = \min\{\theta^{ij}(0,x_j), \theta^{ij}(1,x_j)\}$, redefine

$$\theta^{ij}(0,x_j) \leftarrow \theta^{ij}(0,x_j) - \epsilon \tag{2.90a}$$
$$\theta^{ij}(1,x_j) \leftarrow \theta^{ij}(1,x_j) - \epsilon \tag{2.90b}$$
$$\theta^{j}(x_j) \leftarrow \theta^{j}(x_j) + \epsilon. \tag{2.90c}$$

2. For all $i \in \mathcal{V}$ and $\epsilon = \min\{\theta^i(x_i) \,|\, x_i \in \{0,1\}\}$ redefine

$$\theta^{i} \leftarrow \theta^{i} - \epsilon \tag{2.91a}$$
$$\text{const} \leftarrow \text{const} + \epsilon. \tag{2.91b}$$

After the energy function was brought into normal form, we have either $\theta^{ij}(0,0) = \theta^{ij}(1,1) = 0$ or $\theta^{ij}(0,1) = \theta^{ij}(1,0) = 0$.

We state here the min-cut max-flow theorem which was independently proven in [FF56] and [EFS56].

Theorem 2.4.2. ***Min-cut max-flow theorem:*** *In any network flow with source s and sink t, the maximum flow is equal to the capacity of the minimum cut.*

Proof. See [FF56, Theorem 1]. □

Graph cut algorithms solve the labeling problem by solving the min-cut problem for which algorithms with polynomial complexity are known in the case of positive weights. After computing the min-cut all nodes are divided into two subsets, where nodes in one subset are labeled with 0 and with 1 in the other. Before we detail on

the algorithm let us discuss the question how to construct the corresponding directed graph for the energy function (2.85).

Constructing the Corresponding Directed Graph for the Energy Function in (2.85) We describe how the binary labeling problem can be reformulated as a graph cut problem. Let us assume that $\theta^i(x_i) \geq 0, \forall i \in \mathcal{V}$ and $\theta^{ij}(x_{ij}) \geq 0, \forall ij \in \mathcal{E}$. First the node set is extended by the terminal nodes $\hat{\mathcal{V}} = \mathcal{V} \cup \{s, t\}$ and the edge set is augmented as $\hat{\mathcal{E}} = \mathcal{E} \cup \{si \,|\, i \in \mathcal{V}\} \cup \{it \,|\, i \in \mathcal{V}\}$. For every edge from $\hat{\mathcal{E}}$, a weight is assigned as following: $w_{si} = \theta^i(1)$, $w_{it} = \theta^i(0)$, $w_{ij} = \theta^{ij}(0, 1)$. Without loss of generality, in the following we assume s and t to be in the node set with assigned 0 and 1, respectively. In Fig. 2.2 we illustrate a graph with its minimum cut for a graph with 9 nodes and up to 4 neighboring nodes, as well as an example for a cut.

In [KZ04] it was shown that the energy minimization problem can be transformed into a min-cut problem with non-negative weights if and only if the energy function is submodular.

However even if the function does not satisfy the submodularity condition a partially global solution can be found using a graph-cut-like algorithm, namely quadratic pseudo boolean optimization (QPBO), first proposed in [HHS84] and revised in [KR07].

Quadratic Pseudo Boolean Optimization (QPBO) The QPBO method does not require a submodular function. However, when the function is not submodular, there is no guarantee the solution will be complete, meaning there can be some state variables left with no values assigned.

Let x be a labeling produced by QPBO. Then we can state the following important properties of the QPBO method:

(1) ***If all terms of the energy are submodular, then the algorithm outputs a complete labeling.***

(2) ***Persistency (Weak Autarky):*** Let y be a complete labeling and let z be the combined labeling of x and y, that is $z_i = x_i$ if x_i was assigned a label $\{0, 1\}$ and $z_i = y_i$, otherwise. Then $E(z) \leq E(y)$ holds.

(3) ***Partial optimality (Weak Persistency):*** There exists a labeling x^* minimizing the energy globally such that $x_i^* = x_i$, $\forall i \in \mathcal{V}$ which get a label.

(4) ***Invariance to flipping:*** QPBO is invariant to flipping the values of variables $x_i \in U$ for a subset $\mathcal{U} \subset \mathcal{V}$, interchanging 0 and 1 labels.

In view of property (2) we can see that for an incomplete labeling provided by QPBO, the energy will not increase no matter what the unassigned variables are fixed to. Concerning (3) we remark that there is a set of global optimal solutions to the energy. The reason is the relaxation and the constant term in the energy. (4) is an important property in the case when after flipping the values 0 and 1 from some subset $\mathcal{U}$ of the node set $\mathcal{V}$ would result into submodular energy. Then as a result of (1) the variables of $\mathcal{U}$ will be all labeled.

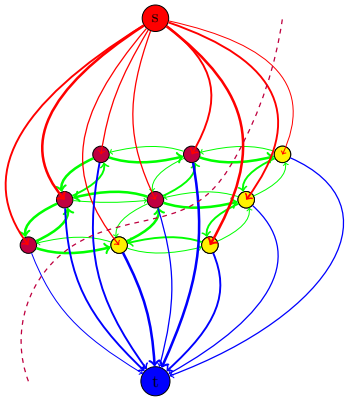


Figure 2.2. - Minimum cut for a graph with 9 nodes and up to 4 neighbors per node. Each node is connected to its neighbors and to the source and the sink nodes. Weight of an edge is represented with its thickness. Thicker edge means bigger weight. After the minimum cut is found the nodes are divided into two subsets, purple and yellow.

Description of the QPBO Algorithm QPBO works like other graph cut algorithms, i.e. it first constructs the graph corresponding to the energy function and then a minimum cut is computed assigning binary labels to all nodes. Any binary energy function can be minimized with QPBO. After the energy is reparametrized with converting all unary and pairwise weights into non-negative ones, the graph corresponding to the energy function, is constructed with a doubled set of nodes. The transformation of the energy to the directed graph is in a similar way as we described for graph cuts, however now for each node $i \in \mathcal{V}$ we introduce a further node $\hat{i}$. The new set of nodes is now defined as $\hat{\mathcal{V}} := \mathcal{V} \cup \{\hat{i} \,|\, i \in \mathcal{V}\} \cup \{s, t\}$. The new edge set is $\hat{\mathcal{E}} := \mathcal{E} \cup \{si, s\hat{i} | i \in \mathcal{V}\} \cup \{it, \hat{i}t \,|\, i \in \mathcal{V}\} \cup \{i\hat{j}, \hat{i}\hat{j}, \hat{i}j \,|\, ij \in \mathcal{E}\}$. The weights assigned to the new edges are defined as

$$
\begin{aligned}
w_{si} = w_{\hat{i}t} &= \frac{1}{2}\theta^i(1) & \qquad w_{it} = w_{s\hat{i}} &= \frac{1}{2}\theta^i(0) \\
w_{ij} = w_{\hat{j}\hat{i}} &= \frac{1}{2}\theta^{ij}(0,1) & \qquad w_{ji} = w_{\hat{i}\hat{j}} &= \frac{1}{2}\theta^{ij}(1,0) \\
w_{i\hat{j}} = w_{j\hat{i}} &= \frac{1}{2}\theta^{ij}(0,0) & \qquad w_{\hat{j}\hat{i}} = w_{\hat{i}j} &= \frac{1}{2}\theta^{ij}(0,0).
\end{aligned}
$$

After finding the minimal cut, the graph is divided into two subsets $\mathcal{V}_s$, and $\mathcal{V}_t$, and the nodes are labeled in the following way

$$
\begin{cases}
x_i = 0 & \text{if } i \in \mathcal{V}_s \text{ and } \hat{i} \in \mathcal{V}_t \\
x_i = 1 & \text{if } i \in \mathcal{V}_t \text{ and } \hat{i} \in \mathcal{V}_s \\
x_i \text{ gets no label} & \text{otherwise .}
\end{cases} \tag{2.92}
$$

The algorithm first finds the minimum cut on the part of the graph with submodular energy terms. The difference of QPBO to other graph cut algorithms is that it selects the cut from the set of all minimum cuts which maximizes the number of nodes

which were assigned a label.

We point out that there are extensions of QPBO to the multi-label case [WIC12]. There are also other graph cut algorithms which do not require submodular functions and can be used for the multi-label case problem, namely move-making algorithms, the $\alpha - \beta$ swap and $\alpha-$expansion algorithm introduced in [BVZ01]. These graph cut approaches iteratively apply the binary graph cut algorithm. Other algorithms for the multi-label case require some other constraints on the smoothness energy term, like convexity [Ish03]. Important to note is the close connection of submodularity and convexity [Lov83].

2.4.5. Potts Model for Segmentation

We give an example of approximate inference for the segmentation problem when using a Potts model.

Image segmentation is separating objects from background and assigning pixels which belong to the same object the same label. There are many ways of grouping objects sharing specific characteristic important for the related analysis. Segmentation is a supervised type of grouping, when the labels are known beforehand. There are also semi-supervised type of clustering techniques which partitions data into a predetermined number of clusters based on similarities. For instance k-means clustering [Llo82] for a fixed number of clusters is a semi-supervised type of clustering. Unsupervised clustering on the other hand is when both the labels and the number of clusters are not known before. This is an ill-posed problem in general and some constraint on the number of clusters has to be provided. Spectral clustering [Lux07, NJW01] can be considered as an unsupervised type of clustering which is based on minimum cut techniques and analyzes a similarity matrix. More precisely, the number of clusters is determined based on the eigenvalues of the similarity matrix.

One way of performing supervised segmentation is using energy minimization methods. The Potts model, although a very old model [Pot52] is broadly used in computer vision for image segmentation. It is a generalization of the Ising model, which is defined on two labels. It acts as prior to cope with the ill-posedness of the segmentation problem. The Potts prior is the constraint that neighboring pixels are in the same region.

The Potts model aims to assign each pixel represented by nodes $i \in \mathcal{V}$ to a region such that an energy function is minimized:

$$E_\theta(x) = \sum_{i \in \mathcal{V}} \theta^i(x_i) + \sum_{ij \in \mathcal{E}} \theta^{ij}[x_i \neq x_j]. \tag{2.93}$$

The first part penalizes pixel assignments e.g. based on the color, while the second is the Potts prior defined on the graph edges $ij \in \mathcal{E}$. Here θ^{ij} are non-negative constants and [.] is the Iverson bracket which is 1 if the expression in the bracket is true and 0, otherwise, such that assignments with different neighbouring labels are penalized.

The Potts function in general can be minimized using approximate inference techniques. It can be easily checked that the Potts function with two labels is a

submodular function, whereas this is not the case for three or more labels. So in the case of two labels the Potts function can be globally optimized with graph cuts for instance. For the case of more labels, minimizing the energy function is an NP-hard problem in $2D$ (not in $1D$) which comes from its equivalence to the multiway cut problem [BVZ01], a generalization of the minimum cut problem. This equivalence was shown in [BVZ98] from which carries over to the use of approximate methods for the multiway cut problem to minimizing the Potts function. For the multi-label case approximate solution can be obtained with graph cuts, $\alpha - \beta$ swap and $\alpha-$expansion [BVZ01], and some LP relaxations [KT07, WJW02].

2.5. Learning

Learning is the dual problem to inference. In inference we saw that given the canonical parameters θ we infer the mean parameters μ. In learning we have given the mean parameters μ which represent a subset of our data, the training data, from which we learn the canonical parameters θ, also called model parameters. Just as inference, learning is a non-trivial problem, however, there are efficient approximations for learning as well.

There are different approaches to learning. The most natural way to solve the learning problem is with MLE as we saw before, see (2.70). Due to the close connection of MLE to Kullback-Leibler (KL) divergence [KL51] the learning problem has been also formulated in other ways.

Depending on how the objective function of the learning problem is formulated, (approximate) learning can be divided into two classes: probabilistic parameter learning which is MLE with different type of approximations, and loss minimizing parameter learning, when the learning problem is defined as minimizing a loss function defined as the cost of predicting approximate mean or canonical parameters on data on which the correct mean parameters are given. Basically the loss function measures some distance between predicted and correct mean or canonical parameters.

In the following sections we describe these two types of learning.

Many other approaches not discussed fall into this categorization. For more on this subject we refer to [NL11].

2.5.1. Probabilistic Parameter Learning

Given empirical mean parameters $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) \in \mathcal{M}$ the problem of learning canonical parameters, or probabilistic learning, is concerned with the problem

$$\sup_{\theta \in \Theta} \langle \theta, \hat{\mu} \rangle - A(\theta). \tag{2.94}$$

The equation above is $A^*(\mu)$, the conjugate dual to the log-partition function which is finite for mean parameters $\mu \in \mathcal{M}$ due to the result of Theorem 2.3.6. Solving the learning problem or optimizing (2.94) is equivalent to computing ∇A^*. This is computationally intractable in the general case. The reason is the high complexity of the marginal polytope space parameters as well as of the lack of a closed form

expression of A^* in the general case. Exceptions are tree structured graphs where junction tree theory applies, see [Lau96]. In the case of tree structured graphs the log-partition function can be written in closed form. When the graph is a tree its probability function can be decomposed as

$$p_{\theta(\mu)}(x) := \prod_{i\in\mathcal{V}} \mu_i(x_i) \prod_{ij\in\mathcal{E}} \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)}, \tag{2.95}$$

a result from the junction tree theory. When the probability can be decomposed as above A^* can be written in closed form. Using this probability distribution decomposition rule and the result from 2.3.6, that A^* is equivalent to the negative entropy for $\mu \in \operatorname{rint} \mathcal{M}$, we have

$$\begin{aligned} A^*(\mu) &= -H(p_{\theta(\mu)}) = -\mathbb{E}_{p_{\theta(\mu)}}[\log p_{\theta(\mu)}(X)] && \text{(2.96a)} \\ &= -\sum_{i\in\mathcal{V}} H_i(\mu_i) + \sum_{ij\in\mathcal{E}} I_{ij}(\mu_{ij}) && \text{(2.96b)} \end{aligned}$$

where the singleton entropy $H_i(\mu_i) := -\sum_{x_i} \mu_i(x_i) \log \mu_i(x_i)$, for $i \in \mathcal{V}$ and the pairwise information $I_{ij} := \sum_{x_i x_j} \mu_{ij}(x_i, x_j) \log \frac{\mu_{ij}(x_i,x_j)}{\mu_i(x_i)\mu_j(x_j)}$.

For general graphs however due to intractability, approximations have been proposed. When relaxing the inference problem we introduced the approximation of the marginal polytope $\mathcal{M}(\mathcal{G})$ with a local polytope $\mathcal{L}(\mathcal{G})$, which is tight for trees. For the learning problem in addition to relaxing the marginal polytope, the intractable $A(\theta)$ should be approximated.

In [WJW05a] a strictly convex twice differentiable approximation to A^* was designed so that the domain of the approximation of A^* is contained in the local polytope relaxation of the marginal polytope. Let us denote with B^* the approximation to $-A^*$. The motivation for constructing the approximate function comes from the closed form of A^* on trees. When the general graph can be considered as a combination of tree graphs, then due to (2.96) the functions A^* on each of the trees would have a closed form expression. In fact any connected graph can be decomposed into spanning trees. Considering a spanning tree decomposition of the connected graph $\mathcal{G}$ into trees $T \in \mathcal{T}$, A^* can be considered as a decomposition of A^* functions in closed form on spanning trees composing the whole graph while weighting every spanning tree T by a probability $P(T) \in [0, 1]$. Using (2.96) the approximation proposed by Wainwright et al. [WJW05a] also known as ***convexified Bethe entropy approximation*** is given by

$$B^* := \sum_{T\in\mathcal{T}} P(T)\Big(\sum_{i\in\mathcal{V}} H_i(\mu_i) - \sum_{ij\in\mathcal{E}(T)} I_{ij}(\mu_{ij})\Big) = \sum_{i\in\mathcal{V}} H_i(\mu_i) - \sum_{ij\in\mathcal{E}(T)} P_{ij} I_{ij}(\mu_{ij}), \tag{2.97}$$

where $P_{ij} = \sum_T P(T) I_T$, also called edge appearance probability, and I_T being the indicator set function, see (2.16). When $P_{ij} > 0$ for all $ij \in \mathcal{E}(T)$, the approximation B^* is strictly convex. Using this result the log-partition function A when expressed as in (2.66) can be approximated using the approximation to $-A^*$, B^* and the relaxed

marginal polytope, $\mathcal{L}_\mathcal{G}$. We denote the approximation to A as B defined by

$$B := \max_{\mu \in \mathcal{L}(\mathcal{G})} (\langle \theta, \mu \rangle + B^*(\mu)). \tag{2.98}$$

It can be shown that when the approximation B^* is strictly convex and twice continuously differentiable, the approximation B is convex and differentiable and has a unique solution, see [Wai06]. This approximation of the log-partition function is used in approximate methods for computing marginals, in particular in tree reweighted message passing methods (TRW) [WJW05b]. We note that loopy belief propagation methods use the normal Bethe approximation which differes from the convexified version in that the weighting probability P from (2.97) is excluded [YFW05].

Moreover, a learning method which uses a strongly convex approximation to $-A^*$, (the entropy) was proven to be globally stable [Wai06].

In practice learning is used to predict mean parameters on novel data. First canonical parameters from given empirical mean parameters are learned, for instance with an approximation of the intractable MLE (2.70). Using the introduced approximation to the log-partition function as in [WJW05a] and given empirical mean parameters of given data samples $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} \phi(X_i)$, approximate MLE, also known as ***surrogate likelihood*** can be computed by maximizing

$$l_B(\theta) := \langle \theta, \hat{\mu} \rangle - B(\theta). \tag{2.99}$$

Optionally a regularizer term $R(\theta)$ can be added and weighted by some $\lambda > 0$ to trade off between the data term and the regularizer. Maximizing the surrogate likelihood can be achieved simply with gradient descent. In the following we call this step perturbation as the canonical parameters can be thought as perturbed exact parameters. Based on the learned approximate parameters and given noisy observations the canonical parameters can be fitted to the observed data, in order to compute new predicted parameters. We call this step fitting or prediction. Perturbation and fitting can be done jointly in one step or separately. Finally, using the predicted canonical parameters mean parameters can be computed in an inference step.

Wainwright et al. proved in [Wai06] that it is of great importance to use the same approximate method for computing the approximate parameters and when inferring the mean parameters in the last step. In particular when learning with approximate MLE the same approximation of the log-partition function should be used for the approximate MLE and the computation of the mean parameters (marginals). Furthermore, Wainwright et al. showed that learning benefits from approximations (when the same type of approximation is used for learning and inference) since in practice the errors introduced by the inexact canonical parameters are partly compensated by the error of the inexact inference.

Learning with approximate MLE can lead to good predictions when the training data is big, as we will see in the following. To this end we interpret MLE as the Kullback-Leibler (KL) divergence [KL51] which is a measure of distance between

probability distributions and defined by

$$\mathrm{KL}(p_{\theta^*}||p_\theta) = \sum_i p_{\theta^*}(x_i) \log \frac{p_{\theta^*}(x_i)}{p_\theta(x_i)}. \tag{2.100}$$

Let θ^* be the parameters corresponding to the given empirical mean parameters now denoted by μ^* and let θ be the parameter we obtain from approximate MLE learning. Then minimizing the KL divergence (2.100) between the two distributions we get

$$\min_{\theta^*} \mathrm{KL}(p_{\theta^*}||p_\theta) = \sum_i p_{\theta^*}(x_i) \log p_{\theta^*}(x_i) - \sum_i p_{\theta^*}(x_i) \log p_\theta(x_i) \tag{2.101a}$$

$$= const - \sum_i p_{\theta^*}(x_i) \log p_\theta(x_i) = \tag{2.101b}$$

$$= -\mathbb{E}_{p_{\theta^*}}[\log p_\theta]. \tag{2.101c}$$

If we consider the MLE as defined in (2.70), then minimizing the negative MLE

$$-\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i) \tag{2.102}$$

converges to the expectation $-\mathbb{E}[\log p_\theta]$ due to the strong law of large numbers, see [Ete81].

That is for very large training data ($n \to \infty$) minimizing the negative MLE is equivalent to minimizing the Kullback-Leibler divergence between the true (or exact) distribution and the searched canonical parameters. This insight motivates the definition of learning as minimization of a distance between true and estimated parameters. However, in practice we are not always given a lot of data in which case learning with MLE performs poorly.

2.5.2. Loss Minimizing Parameter Learning

In the machine learning community learning is usually defined as the minimization of a loss function. We give a brief description of learning from this point of view. The defined objective loss function measures a distance between the predicted approximate mean parameters and the given correct mean parameters, the empirical mean parameters. As described in the previous subsection, one way is to learn the canonical parameters using approximate MLE from empirical mean parameters. Based on that result, in the inference step the predicted mean parameters can be computed.

In other words, learning in graphical models is concerned with building a graphical model that describes the problem we want to solve. In practice learning methods minimize a loss function which measures the similarity between the model we want to obtain and the model that corresponds to the observed data. Interpreted differently, the loss function evaluates the distance Δ between ground truth μ^* and the minimum

energy configuration of the MRF model. This can be written as

$$\min_{\theta} \Delta(\mu(\theta), \mu^*) \quad \text{subject to} \quad \mu(\theta) = \arg\min_{\theta} E_\theta(\mu), \tag{2.103}$$

with the energy function defined by

$$E_\theta(\mu) = \sum_{i \in \mathcal{V}} \theta^i(x_i)\mu_i(x_i) + \sum_{ij \in \mathcal{V}} \theta^{ij}(x_{ij})\mu_{ij}(x_{ij}). \tag{2.104}$$

In this work we concentrate on supervised learning, i.e. ground truth is provided for the training data, based on what we have to predict output for new observed test data. We refer to the learning problem to be unsupervised or semi-supervised if ground truth is not or only partially available for the training data.

This thesis is concerned with the problem of learning for graphical models. We introduce 2 novel methods for loss minimizing parameter learning which we will describe in Chapter 4.

One of our novel learning methods is based on inverse linear programming, which we will briefly introduce in the next section.

2.6. Inverse Linear Programming

Given some observation, the aim of solving an inverse problem is to determine the factors that produce them. For instance computer tomography solves an inverse problem for reconstructing a physical volume for which only some measurements were observed.

Similarly in inverse optimization, given the observed optimal solution one computes the model parameters which would result in the optimal solution. In this sense one can think about learning as an inverse problem. Given observations (mean parameters, ground truth data) we want to compute model parameters (canonical parameters) that lead to the given observations, when solving an inference problem.

In inverse optimization a feasible solution to the inverse problem may be found which is not an optimal one, however. The optimal one can be very difficult or impossible to find only based on the observed solution. In order to find the optimal model parameters that produced the observed solution, the given feasible parameters should be perturbed as little as possible so that they lead (correspond) to an optimal solution. This is what we refer to by inverse programming or inverse linear programming when the problem to solve is a linear program (LP) as it is in our case when applied to the learning problem. More precisely we define inverse linear programming as in [ZL96] as following:

Definition 2.6.1. Let the linear program (LP) be given by

$$\min_{x} \langle \hat{c}, x \rangle \quad \text{s.t. } Ax \geq b \text{ and } l \leq x \leq u \tag{2.105}$$

where $\hat{c}, x, l, u \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$. Let $\hat{x}$ be a feasible solution to the given LP. We want to find $\tilde{c}$ which is as close as possible to $\hat{c}$ so that the feasible solution $\hat{x}$

becomes an optimal solution to the adjusted LP given by

$$\min_x \langle \tilde{c}, x \rangle, \quad \text{s.t. } Ax \geq b \text{ and } l \leq x \leq u \tag{2.106}$$

whose solution is an optimal x. Then the inverse LP is expressed as

$$\min ||\tilde{c} - \hat{c}||_1, \quad \text{s.t. } x \text{ is an optimal solution to (2.106)} . \tag{2.107}$$

Note that for the definition above we have to know the optimal solution x in order to solve the inverse LP. This definition for the ℓ_1 norm defined by $||\tilde{c} - \hat{c}||_1 = \sum_i |\tilde{c}_i - \hat{c}_i|$ was extended to the ℓ_∞ norm defined by $||\tilde{c} - \hat{c}||_\infty = \max_i |\tilde{c}_i - \hat{c}_i|$ in the later paper by Zhang [ZL99] and to the weighted case (weighted norm) in [AO01]. What is important is that whenever the original problem is an LP the inverse problem is an LP, too [AO01]. The feasible region of the inverse problem is formulated using the complementary slackness constraints and the constraints of the dual problem.

Next we derive the inverse linear program given the original LP.

Given the primal LP as in (2.105) we first define its dual

$$\max_y \langle b, y \rangle + \langle l, \lambda \rangle - \langle u, \psi \rangle \tag{2.108a}$$

$$\text{s.t. } Ay + \lambda - \psi = \hat{c}, \quad y \geq 0, \lambda \geq 0, \psi \geq 0 \tag{2.108b}$$

where y is the associated dual variable to the constraint on x expressed by the matrix A. Furthermore, λ and ψ are the dual variables associated with the constraints on x, l, and u respectively. Linear programming optimality conditions state that the primal and dual solutions x and y are optimal if they are both feasible for the corresponding problems and the complementary slackness conditions are satisfied, i.e.

$$\text{a) } Ax > b \implies y = 0, \tag{2.109a}$$

$$\text{b) } x > l \implies \lambda = 0, \text{ and} \tag{2.109b}$$

$$\text{c) } x < u \implies \psi = 0. \tag{2.109c}$$

As stated in Definition 2.6.1 we want $\hat{x}$ to be an optimal solution to the perturbed problem (2.106). We can consider the primal and the dual for the perturbed problem as the primal and the dual of the original problem just with $\hat{c}$ replaced by $\tilde{c}$. We call the primal and the dual of the perturbed problem the primal perturbed and the dual perturbed. Now, due to the optimality conditions, $\hat{x}$ is an optimal solution to the perturbed primal if and only if there exists a dual problem to the perturbed primal and the complementary slackness conditions as in (2.109) are satisfied. From the constraints of the dual problem (2.108) and the complementary slackness conditions

we can define the following index sets for $\hat{x}$

$$\mathcal{B} := \{i \in [m] : (A\hat{x} - b)_i = 0\} \tag{2.110a}$$
$$\mathcal{L} := \{j \in [n] : (\hat{x} - l)_j = 0\} \tag{2.110b}$$
$$\mathcal{U} := \{j \in [n] : (\hat{x} - u)_j = 0\} \tag{2.110c}$$
$$\mathcal{S} := \{j \in [n] : 0 < \hat{x}_j < u_j\}, \tag{2.110d}$$

then the constraints in the perturbed dual can be written as

$$(Ay + \lambda - \tilde{c})_{\mathcal{L}} = 0 \tag{2.111a}$$
$$(Ay - \psi - \tilde{c})_{\mathcal{U}} = 0 \tag{2.111b}$$
$$(Ay - \tilde{c})_{\mathcal{S}} = 0 \tag{2.111c}$$
$$y_{\mathcal{B}} \geq 0, \lambda_{\mathcal{L}} \geq 0, \psi_{\mathcal{U}} \geq 0. \tag{2.111d}$$

Let us denote $c = \tilde{c} - \hat{c}$. Then the inverse problem as defined in (2.107) is to minimize the ℓ_1 norm of c such that the constraints as defined in (2.111) are satisfied. To this end, let us write the ℓ_1 norm as $||c||_1 = c^+ + c^-$, where $c^+ = \max\{c, 0\}$ and $c^- = -\min\{c, 0\}$. Then we can define the inverse LP by the constraints (2.111) where instead of $\tilde{c}$ we use $\tilde{c} = \hat{c} + c = \hat{c} + c^+ - c^-$

$$\textbf{Inverse LP}: \min_{c^+, c^- \geq 0} \langle \mathbb{1}, c^+ + c^- \rangle \text{ s.t.} \tag{2.112a}$$
$$(Ay - c^+ + c^- + \lambda - \hat{c})_{\mathcal{L}} = 0 \tag{2.112b}$$
$$(Ay - c^+ + c^- - \psi - \hat{c})_{\mathcal{U}} = 0 \tag{2.112c}$$
$$(Ay - c^+ + c^- - \hat{c})_{\mathcal{S}} = 0 \tag{2.112d}$$
$$y_{\mathcal{B}} \geq 0, \lambda_{\mathcal{L}} \geq 0, \psi_{\mathcal{U}} \geq 0. \tag{2.112e}$$

The problem is clearly an LP, which brings us to the conclusion that the inverse of an LP is an LP too. In fact the inverse problems of some LPs originating from the minimum cut, the assignment problem and the shortest path problem have their inverse problems with the ℓ_1 norm as an LP of the same kind as the original problem.

3. Metric Learning for Segmentation

3.1. Introduction

When performing segmentation or clustering, one usually clusters data using some notion of similarity. The choice of the similarity measure is involved and strongly problem-specific and influences the quality of the clustering considerably. One approach is to determine a similarity measure based on a clustered training data by minimizing an appropriate energy function. This is known as ***metric learning***. The learned metric can then be used in a k-means clustering algorithm [Llo82].

One way to think about metric learning is to consider that we want to learn some transformation of the Euclidean distance $d(x, y) = ||x - y||_2$, into $d(f(x), f(y)) = ||f(x) - f(y)||_2$. Depending on the function f this transformation can be ***linear***, if $f(x) = Mx$ for some matrix $M \in \mathbb{R}^{|\dim(x)| \times |\dim(x)|}$, or ***nonlinear***, if $f(x)$ is a nonlinear function. Please note that the data x and y are some prespecified feature vectors, defined depending on the type of the problem and data. Due to this, the distance metric we learn is a transformation of the feature distance.

Let us assume the data is given as a collection of feature vectors $X = \{x_i\}_{i \in \mathcal{V}} \subset \mathbb{R}^n$, indexed by the vertices of a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Two subsets defined on pairs of the vectors of X are classified as

$$\mathcal{D} \subset \mathcal{E}: \quad \text{indices of pairwise dissimilar feature vectors,} \tag{3.1a}$$

$$\mathcal{S} \subset \mathcal{E}: \quad \text{indices of pairwise similar feature vectors} \tag{3.1b}$$

and in addition a third set of triplets can be given defined by

$$\mathcal{R} = \{(i, j, k) \,|\, x_i \text{ is more similar to } x_j \text{ than to } x_k\}. \tag{3.2}$$

Then the optimization problem of a feature transformation function $f(x) = Mx$ is given by

$$\min_{M \in \mathrm{dom}(M)} l(M, \mathcal{D}, \mathcal{S}, \mathcal{R}) + \delta r(M). \tag{3.3}$$

Here $l(M, \mathcal{D}, \mathcal{S}, \mathcal{R})$ is a loss function which is penalized when the constraint sets are violated, $r(M)$ is a regularizer and $\delta \geq 0$ a parameter controlling the trade off between the loss and the regularizer. A common choice also employed in this work is $\mathrm{dom}(M) = \mathbb{S}_+^n$ is the space of symmetric positive semi-definite matrices. In some applications, the set is further restricted for instance to symmetric positive semi-definite matrices with bounded trace. Sometimes the metric learning optimization

problem can be written using constraints on the loss function

$$\min_{M \in \mathrm{dom}(M)} r(M) \tag{3.4a}$$

$$\text{s.t. } l(M, \mathcal{D}, \mathcal{S}, \mathcal{R}) \leq c \tag{3.4b}$$

where c is some constant. We point out the resemblance of the metric learning problem as formulated in (3.3) and (3.4) to support vector machine (SVM) [SS01] where also a loss function is minimized, however w.r.t. the parameters of a separating plane instead of a transformation matrix.

Depending on the type of the data we have, metric learning can be ***supervised***, when the data is associated with labels from which the subsets $\mathcal{S}, \mathcal{D}, \mathcal{R}$ as defined in (3.1) and (3.2)) can be derived. In the case of ***weakly supervised*** learning, the data is provided with no labels but only grouped into similarity/dissimilarity subsets $\mathcal{S}, \mathcal{D}, \mathcal{R}$. ***Semi-supervised*** learning refers to data classified into groups $\mathcal{S}, \mathcal{D}, \mathcal{R}$ and additionally for some of the training data labels are available. ***Unsupervised*** metric learning refers to the case when no prior information is given, that is the training data is not grouped into subsets $\mathcal{S}, \mathcal{D}$ and $\mathcal{R}$, and labels are not provided. This is clearly the most difficult type of metric learning and there has not been much research in this direction. We will later discuss dimensionality reduction methods which can also be interpreted as unsupervised metric learning methods.

Metric learning methods can be distinguished depending on whether they are ***global***, i.e. when a single metric is learned taking into account all training data points, or ***local***, when multiple metrics are learned on data subsets. Depending on the problem and data type a local or a global strategy is beneficial. Local metrics for instance perform better when the data is heterogeneous, since then on each homogeneous subpart of the image a local metric is learned, while a single global metric might not be flexible enough to represent complex data sets.

In nonlinear metric learning methods the feature vectors x_i are transformed by a nonlinear function $f(x_i)$. Learning this function usually involves a more difficult optimization problem which generally is not convex. The simplest way of obtaining nonlinear metric is with kernelizing a linear metric. Nonlinear methods usually work better for clustering the training dataset but there is no guarantee that they generalize better to the testing dataset.

In this work we concentrate on linear, global and supervised metric learning methods. The most prominent type of metric learned is a Mahalanobis metric. We first start with illustrating a Mahalanobis distance metric learning and continue with the most influential works on metric learning as well as their optimization procedures. We conclude by proposing a different optimization technique to two existent metric learning methods and illustrate its performance on some experiments.

3.2. Mahalanobis Distance Metric Learning

Let us first start with the definition of a metric.

Definition 3.2.1. A ***metric*** or a distance function d on a set X has to satisfy the following properties, $\forall x_i, x_j, x_k \in X$:

$$\begin{aligned} &1. \quad d(x_i, x_j) + d(x_j, x_k) \geq d(x_i, x_k) \text{ (triangular inequality)} \\ &2. \quad d(x_i, x_j) \geq 0 \text{ (nonnegativity)} \\ &3. \quad d(x_i, x_j) = d(x_j, x_i) \text{ (symmetry)} \\ &4. \quad d(x_i, x_j) = 0 \iff x_i = x_j \text{ (distinguishability).} \end{aligned} \tag{3.5}$$

A function d which satisfies only the first three properties is called a ***pseudo-metric***.

Metric learning is also called ***Mahalanobis metric learning*** as it employs a ***Mahalanobis distance function*** [Mah36] which is defined as:

$$d_{\text{Mahal}} \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \qquad (x_i, x_j) \mapsto d_{\text{Mahal}}(x_i, x_j) := \langle (x_i - x_j), \Sigma^{-1}(x_i - x_j) \rangle^{1/2} \tag{3.6}$$

where Σ is a covariance matrix defined on the data. When the data fits a Gaussian distribution the covariance matrix can be a good choice for a distance matrix. However, the complexity of computing the covariance matrix is cubic in the number of dimensions. In fact, in the literature the term Mahalanobis distance usually refers to a distance where a different matrix M is used instead of the usually unknown inverse covariance matrix Σ^{-1}, i.e.:

$$d_M(x_i, x_j) := \langle (x_i - x_j), M(x_i - x_j) \rangle^{1/2}. \tag{3.7}$$

As d_M should be real-valued, the matrix M is constrained to be positive semi-definite, i.e. $M \in \mathbb{S}_+^n$, see (2.8) for a definition. The Mahalanobis metric (3.7) is a pseudo-metric, as it satisfies the first three properties in (3.5). In slight abuse of terminology by metric we will also refer to pseudo-metric in the following. We remark that sometimes in metric learning algorithms the squared distance is considered instead, which is not a pseudo-metric itself, but considering the squared distance it eases the optimization process, due to the beneficial linearity in M.

Metric learning methods aim at learning a distance d_M defined as in (3.7), which separates the data with indices in the set $\mathcal{D}$ from the data with indices in the set $\mathcal{S}$, with $\mathcal{S}$ and $\mathcal{D}$ as defined in (3.1), in the best possible way. Then the problem amounts to moving feature vector pairs with indices from $\mathcal{S}$ close together while simultaneously separating feature vector pairs with indices from $\mathcal{D}$, both assessed using the distance measure.

In the rest of this chapter we denote the identity matrix with I. Clearly when $M = I$, we have the Euclidean distance. Since $M \in \mathbb{S}_+^n$, it has nonnegative eigenvalues and utilizing the eigenvalue decomposition we have $M = L^\top L$, with $L \in \mathbb{R}^{l \times n}$, $l = \operatorname{rank}(M)$. Then

$$\begin{aligned} d_M(x_i, x_j) &= \langle x_i - x_j, L^\top L(x_i - x_j) \rangle^{1/2} \\ &= \langle Lx_i - Lx_j, Lx_i - Lx_j \rangle^{1/2} \end{aligned} \tag{3.8}$$

that is $x_i, x_j \in \mathbb{R}^n$ are projected to a space with dimension $l \leq n$. In this sense when M is not a full rank matrix, metric learning can be also interpreted as a dimensionality reduction method. For instance principal component analysis (PCA) [SSM98] is a type of a metric learning method which maps data to a lower dimensional space. In addition this can be thought of a type of an unsupervised metric learning method. But as it more focuses to map the training data to a low-dimensional rather than to a highly informative space, it does not generalize well to unseen data.

Mahalanobis metric learning usually leads to a convex optimization problem, allowing to find a globally optimal solution. Usually a projected subgradient method is employed, where the projection is done on the space of symmetric positive semi-definite matrices $\mathbb{S}^n_+$. This requires the computation of eigenvalues of M in order to subsequently replace negative ones by 0. The eigenvalue decomposition is the bottleneck of metric learning approaches as this is computationally expensive for high dimensional M and thus for high dimensional feature vectors. Minimizing the distance in the decomposed form (3.8) leads to an unconstrained optimization problem in L not requiring a projection, however at expense of a non-convex objective function.

In general the metric learning problem is formulated as in (3.3) or (3.4) with M being symmetric positive semi-definite matrix. Metric learning approaches differ with respect to how the loss function and the regularizer are chosen, as well as the optimization procedures proposed. We will give an overview on some of the more successful and influencing metric learning approaches in the next section. We concentrate only on supervised global linear metric learning. The introduced models will be either in the form (3.3) or (3.4).

3.3. Representative Existing Approaches

Metric learning is still a hot topic in research, in particular in computer vision, bioinformatics and information retrieval. Among the many computer vision applications are image classification, face recognition, pose estimation. Many attempts exist to formulate the metric learning problem appropriately. In this work we will discuss only the most influential metric learning approaches, the models proposed therein and the optimization methods utilized. For a more profound overview on metric learning methods we refer to the two most recent survey papers [Kul12, BHS14].

3.3.1. Mahalanobis Metric Learning for Clustering

The first most influencing work on metric learning was the work of Xing et al. [XNJR02] also called Mahalanobis metric learning for clustering. The intention was to improve k-means clustering using a learned appropriate similarity measure or distance metric. Xing et al. formulated the metric learning problem as a convex optimization problem which we will show in Sect. 3.5.1. The authors proposed the

following formulation:

$$\max_{M \succeq 0} \sum_{(i,j)\in\mathcal{D}} d_M(x_i, x_j) \tag{3.9a}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{S}} d_M^2(x_i, x_j) \leq 1, \tag{3.9b}$$

with d_M as defined in (3.7) and the sets $\mathcal{S}$ and $\mathcal{D}$ as defined in (3.1). Constraint (3.9b) is required to prevent M to grow above all limits. Instead of the value 1 in (3.9b) another constant can be taken and this will not change the optimization problem, only the matrix M will be rescaled. The problem formulation aims at learning a suitable matrix M, so that the sum of distances between dissimilar points is maximized while the sum of distances between similar points is bounded from above. In view of the metric learning objective as defined in (3.4) the method of Xing et al. when converted to a minimization problem has a regularizer $r(M) = -\sum_{(i,j)\in\mathcal{D}} \sqrt{\text{tr}(MX_{ij})}$, where $X_{ij} := (x_i - x_j)(x_i - x_j)^\top$ and a constrained loss function $l(M, \mathcal{S}) = \text{tr}(MX_{\mathcal{S}})$ where $X_{\mathcal{S}} = \sum_{(i,j)\in\mathcal{S}} X_{ij}$.

3.3.2. Large-Margin Nearest Neighbors (LMNN) Method

A broadly used supervised metric learning method, closely related to SVM is the ***Large-Margin Nearest Neighbors*** **(LMNN)** method proposed by Weinberger et al. [WBS06, WS08, WS09]. The similarity constraint sets are now defined using local information. LMNN aims at bringing k-nearest neighbors with the same label close in sense of metric distance, while maximizing the separating margin between k-nearest neighbors with different labels. Weinberger et al. defined the nearest neighbor type of metric learning problem in the following way:

$$\min_{M\succeq 0}(1-\mu) \sum_{(i,j)\in\mathcal{S}} d_M^2(x_i, x_j) + \mu \sum_{(i,j,k)\in\mathcal{R}} (1-y_{ik})[1 + d_M^2(x_i, x_j) - d_M^2(x_i, x_k)]_+ \tag{3.10}$$

where the first term is the regularizer $r(M) = \text{tr}(MX_{\mathcal{S}})$, with $X_{\mathcal{S}} = \sum_{(i,j)\in\mathcal{S}}(x_i - x_j)(x_i - x_j)^\top$ and the second term is the hinge loss defined by $[z]_+ = \max\{0, z\}$. Sets $\mathcal{S}$, $\mathcal{D}$ and $\mathcal{R}$ are now defined based on local information, i.e. in (3.1) and (3.2) point pairs (x_i, x_j) with labels y_i and y_j are restricted to those where x_j is in the k-neighborhood of x_i and $y_i = y_j$. Furthermore, y_{ik} denotes an indicator variable such that $y_{ik} = 1$ if $y_i = y_k$ and zero otherwise, $\mu \in [0, 1]$ is a parameter which controls the trade off between separating and bringing data together. In the above defined problem from the first term it is clear that LMNN aims at minimizing distances between similar (local, k-nearest neighbors) pairs of points. As for the second term when $y_{ik} \neq 1$, that is when x_i and x_k are dissimilar, the defined hinge loss is zero when the dissimilar pair (x_i, x_k) has a distance which is by at least 1 bigger than the distance of the more similar pair (x_i, x_j), see definition of (3.2). Or in other words the margin of separation between dissimilar points should be sufficiently big in order to lead to a zero loss.

A method closely related to LMNN was proposed in [NG08] shown to outperform LMNN and SVM as well. The only difference of the method in [NG08] to LMNN is that it is formulated as a quadratic semi-definite programming problem. In addition the authors in [NG08] propose kernelization of the method.

3.3.3. Metric Learning as Eigenvalue Optimization

An important result in the metric learning literature was proven by Ying et al. in [YL12]: starting from the formulation of the metric learning problem as defined in [XNJR02], the authors reformulate it as a convex optimization problem and show the equivalence to minimizing the largest eigenvalue of a symmetric matrix [LO96, Ove88].

In the formulation (3.9) the *sum* of distances of dissimilar pairs of points with indices in $\mathcal{D}$ is maximized. Alternatively, we can maximize the *minimum* of all distances of points with indices in $\mathcal{D}$, while enforcing an upper bound on the distance between similar point pairs with indices in $\mathcal{S}$. Then learning M amounts to solving the convex optimization problem

$$\max_{M \succeq 0} \min_{(i,j)\in\mathcal{D}} d_M^2(x_i, x_j) \tag{3.11a}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{S}} d_M^2(x_i, x_j) \leq 1. \tag{3.11b}$$

Remark 3.3.1. In the formulation of Xing et al. as in (3.9) the distance in the objective is not squared since this would lead always to a rank one matrix M and a projection of the transformed data on a line as shown in [XNJR02]. However, Ying et al. consider the squared distance in [YL12] as this results in a linear objective and it happens to be crucial in the reformulation as an eigenvalue optimization problem of the metric learning problem.

Using the shorthands

$$X_{ij} := (x_i - x_j)(x_i - x_j)^\top, \tag{3.12a}$$

$$X_\mathcal{D} := \sum_{(i,j)\in\mathcal{D}} X_{ij}, \qquad X_\mathcal{S} := \sum_{(i,j)\in\mathcal{S}} X_{ij}, \tag{3.12b}$$

problem (3.11) reads

$$\max_{M \succeq 0} \min_{(i,j)\in\mathcal{D}} \langle X_{ij}, M \rangle \tag{3.13a}$$

$$\text{s.t. } \langle X_\mathcal{S}, M \rangle \leq 1. \tag{3.13b}$$

Please note that with $r(M) = -\min_{(i,j)\in\mathcal{D}} \operatorname{tr}(MX_{ij}) = -\min_{(i,j)\in\mathcal{D}} \langle X_{ij}, M \rangle$ and $l(M, \mathcal{S}) = \operatorname{tr}(MX_\mathcal{S}) = \langle X_\mathcal{S}, M \rangle$ we get the standard formulation as defined in (3.4).

In order to show the equivalence of the metric learning problem (3.13) to the eigenvalue optimization problem we first proof a necessary result for the main proof.

Lemma 3.3.1. *Let M^* be optimal for (3.13), then $\langle X_\mathcal{S}, M^* \rangle = 1$.*

Proof. Suppose M^* is optimal but $\langle X_{\mathcal{S}}, M^* \rangle = \alpha < 1$, and let $k^* = (i^*, j^*)$ denote the index of a minimal $\langle X_k, M \rangle$ that, together with M^*, defines the objective of (3.13). Then, setting $M = \frac{1}{\alpha} M^* \succeq 0$, we get $\langle X_{\mathcal{S}}, M \rangle = 1$ and the objective function value $\min_{(i,j)\in\mathcal{D}} \langle X_{ij}, M \rangle = \frac{1}{\alpha} \langle X_{k^*}, M^* \rangle > \langle X_{k^*}, M^* \rangle$, which contradicts the assumption that M^* is optimal, and so $\langle X_{\mathcal{S}}, M^* \rangle = 1$ as claimed. □

We introduce the linear mappings

$$\mathcal{A} \colon \mathcal{S}^n \to \mathbb{R}^{|\mathcal{D}|}, \qquad S \mapsto (\dots, \langle X_{ij}, S \rangle, \dots)^\top, \tag{3.14a}$$

$$\mathcal{A}^\top \colon \mathbb{R}^{|\mathcal{D}|} \to \mathcal{S}^d, \qquad u \mapsto \sum_{(i,j)\in\mathcal{D}} u_{ij} X_{ij} \tag{3.14b}$$

satisfying

$$\langle \mathcal{A}S, u \rangle = \sum_{(i,j)\in\mathcal{D}} u_{ij} \langle X_{ij}, S \rangle = \sum_{(i,j)\in\mathcal{D}} \langle S, u_{ij} X_{ij} \rangle = \langle S, \mathcal{A}^\top u \rangle, \; \forall S \in \mathcal{S}^d, \; \forall u \in \mathbb{R}^{|\mathcal{D}|}. \tag{3.15}$$

Please note that $u = (..., u_{ij}, ...) \in \mathbb{R}^{|\mathcal{D}|}, (i,j) \in \mathcal{D}$ denotes a vector, where each index ij corresponds to an index pair (i,j) from the set of dissimilar pairs $\mathcal{D}$. Using the defined mappings we can state the main result:

Proposition 3.3.2. *Problem* (3.13) *has the form*

$$\min_{u \in \Delta_{|\mathcal{D}|-1}} \lambda_{\max}(\mathcal{A}^\top u, X_{\mathcal{S}}), \tag{3.16}$$

where with $\Delta_{|\mathcal{D}|-1}$ *we define the* $|\mathcal{D}| - 1$ *dimensional simplex, see 2.2.5.*

The proof of Proposition 3.3.2 below relies on the following variational characterization of the maximum eigenvalue of a symmetric matrix A [OW93]:

$$\lambda_{\max}(A) = \sigma_{\mathcal{C}}(A) = \sup_{S \in \mathcal{C}} \langle A, S \rangle, \qquad \mathcal{C} = \{S \in \mathbb{S}^d_+ \colon \operatorname{tr}(S) = 1\}. \tag{3.17}$$

More generally, for the largest generalized eigenvalue $\lambda_{\max} = \lambda_{\max}(A, B)$ of the matrix pencil (A, B) satisfying

$$Ax = \lambda_{\max} B x, \qquad A \in \mathbb{S}^n, \quad B \in \mathbb{S}^d_+ \tag{3.18}$$

for some positive semi-definite matrix $B \succeq 0$, we have

$$\lambda_{\max}(A, B) = \sigma_{\mathcal{C}_B}(A) = \sup_{\tilde{S} \in \mathcal{C}_B} \langle A, \tilde{S} \rangle, \tag{3.19a}$$

$$\mathcal{C}_B = \{\tilde{S} = Q S Q^\top \colon \operatorname{tr}(S) = 1, \; S \succeq 0, \; Q \in \mathbb{O}^{d \times d}, \; Q^\top B Q = I\}, \tag{3.19b}$$

where $Q \in \mathbb{O}^{d \times d}$ is an orthonormal matrix satisfying the specified condition, where with $\mathbb{O}^{d \times d}$ we denote the set of square matrices with dimension d which have orthonormal columns. Inserting the parametrization of $\tilde{S}$ into the objective and using $\langle A, \tilde{S} \rangle = \operatorname{tr}(A\tilde{S}) = \operatorname{tr}(AQSQ^\top) = \operatorname{tr}(Q^\top A Q S) = \langle Q^\top A Q, S \rangle$, we can rewrite

(3.19) as

$$\lambda_{\max}(A,B) = \sigma_{\mathcal{C}}(Q^\top AQ) = \sup_{S\in\mathcal{C}} \langle Q^\top AQ, S\rangle, \tag{3.20a}$$

$$\mathcal{C} = \{S \in \mathbb{S}_+^d \colon \operatorname{tr}(S) = 1\}, \quad Q \in \mathbb{O}^{d\times d}, \quad Q^\top BQ = I. \tag{3.20b}$$

Proof of Proposition 3.3.2. With $u \in \Delta_{|\mathcal{D}|-1}$, we write

$$\min_{(i,j)\in\mathcal{D}} \langle X_{ij}, M\rangle = \min_{u\in\Delta_{|\mathcal{D}|-1}} \sum_{(i,j)\in\mathcal{D}} u_{ij}\langle X_{ij}, M\rangle = \min_{u\in\Delta_{|\mathcal{D}|-1}} \langle \mathcal{A}^\top u, M\rangle \tag{3.21}$$

and for problem (3.13) in view of Lemma 3.3.1,

$$\max_{M\succeq 0} \min_{(i,j)\in\mathcal{D}} \langle X_{ij}, M\rangle = \max_{M\succeq 0} \min_{u\in\Delta_{|\mathcal{D}|-1}} \sum_{(i,j)\in\mathcal{D}} u_{ij}\langle X_{ij}, M\rangle \tag{3.22a}$$

$$= \max_{M\succeq 0} \min_{u\in\Delta_{|\mathcal{D}|-1}} \langle \mathcal{A}^\top u, M\rangle \quad \text{s.t.} \quad \langle X_{\mathcal{S}}, M\rangle = 1. \tag{3.22b}$$

Because both feasible sets $\mathbb{S}_+^d \cap \{M \in \mathbb{S}^n \colon \langle X_{\mathcal{S}}, M\rangle = 1\}$ for M and $\Delta_{|\mathcal{D}|-1}$ for u, respectively, are compact convex, and since the objective is separately linear in M and u, a saddle point (u^*, M^*) exists (Theorem D.4.2 in [BTN01]). Hence, we can interchange min and max without changing the solution, then (3.22) becomes

$$\min_{u\in\Delta_{|\mathcal{D}|-1}} \max_{M\succeq 0} \langle \mathcal{A}^\top u, M\rangle \tag{3.23a}$$

$$\text{s.t. } \langle X_{\mathcal{S}}, M\rangle = 1. \tag{3.23b}$$

Now, let $Q \in \mathbb{O}^{d\times d}$ be an orthonormal matrix such that

$$Q^\top X_{\mathcal{S}} Q = I. \tag{3.24}$$

Then $\langle X_{\mathcal{S}}, M\rangle = \langle Q^\top X_{\mathcal{S}} Q, Q^\top MQ\rangle = \langle I, Q^\top MQ\rangle = \operatorname{tr}(Q^\top MQ)$. Hence, setting

$$S := Q^\top MQ \tag{3.25}$$

with $Q \in \mathbb{O}^{d\times d}$ satisfying (3.24), we may rewrite (3.23) in the form (3.20),

$$\min_{u\in\Delta_{|\mathcal{D}|-1}} \max_{M\succeq 0} \langle \mathcal{A}^\top u, M\rangle = \min_{u\in\Delta_{|\mathcal{D}|-1}} \max_{S\in\mathcal{C}} \langle Q^\top(\mathcal{A}^\top u)Q, S\rangle \tag{3.26a}$$

$$= \min_{u\in\Delta_{|\mathcal{D}|-1}} \lambda_{\max}(\mathcal{A}^\top u, X_{\mathcal{S}}). \tag{3.26b}$$

□

In addition to this result, in [YL12] the authors also reformulated the LMNN metric learning as an eigenvalue optimization problem.

The equivalent transformation of the metric learning problem (3.11) to an eigenvalue optimization problem, allows to exploit the literature on this subject and optimize with suitable optimization methods the above defined convex and non differ-

entiable problem. In Sect. 3.4.2 we will discuss optimization methods for efficiently solving this convex non differentiable problem.

3.3.4. Online Metric Learning

For large problem size, metric learning algorithms are usually inefficient. The need for reducing computational time or the intractability to access all data at the same time has lead to the development of online metric learning algorithms. However, the time reduction is usually at the expense of finding only an approximate solution instead of an optimal one. As a result online metric learning approaches usually perform worse in practice than batch metric learning methods, but usually come together with an estimate on the error caused by the approximation. We here focus on the formulation of the proposed metric learning problem for the online setting and address the corresponding optimization procedure in Sect. 3.4.3.

We start with the formulation of the supervised online metric learning problem in the general case. Instead of sets $\mathcal{S}$, $\mathcal{D}$ and $\mathcal{R}$ indicating similarity, dissimilarity and relative similarity, respectively, we consider a triplet set $\mathcal{T} = \{x_i, x_i', y_i\}_i$, such that $y_i = 1$ if x_i and x_i' are similar and $y_i = -1$ if they are dissimilar.

The online metric learning problem can then be written as

$$d_M^2(x_i, x_i') \leq b - \frac{\gamma}{2} \qquad \text{if } y_i = 1, \text{ and} \tag{3.27a}$$

$$d_M^2(x_i, x_i') \geq b + \frac{\gamma}{2} \qquad \text{if } y_i = -1 \tag{3.27b}$$

which is equivalent to

$$y_i(b - d_M^2(x_i, x_i')) \geq \gamma \tag{3.28}$$

for $i \in [|\mathcal{T}|]$, γ is the margin of separation, and b some threshold value which serves for predicting similarities between pairs, and w.l.o.g. can be set to 1. The idea is to introduce a loss function that penalizes M for which the predicted similarity of (x_i, x_i') is not consistent with y_i. For this purpose we consider an adapted hinge loss defined on the particular metric learning problem in (3.28) with $b = 1$, given by

$$l_i(M, \gamma) = \max\{0, y_i(d_M^2(x_i, x_i') - 1) + \gamma\}. \tag{3.29}$$

For $b = 1$ and $d_M^2(x_i, x_i') \geq 1$, the point pair is predicted to be dissimilar and similar otherwise. It can be easily checked that l_i vanishes when (3.28) is satisfied and otherwise penalizes the deviation depending on the margin γ. Then the online metric learning problem amounts to finding a sufficiently large margin γ such that the loss (3.29) is minimized. Additionally, often a regularizer is used on the positive semi-definite matrix M in order to exclude unbounded solutions. A regularizer based on the Frobenius norm is given by

$$r(M) = \frac{1}{2}||M||_F^2. \tag{3.30}$$

One of the first most influential works on online metric learning was the one by

Shalev-Shwartz et al. [SSSN04] where an online metric learning approach called ***Pseudo-Metric Online Learning Algorithm*** **(POLA)** was proposed. The objective function involves the adapted hinge loss (3.29), while restricting M to the positive semi-definite cone.

Another online metric learning algorithm was proposed in [DKJ+07], namely ***Information-Theoretic Metric Learning*** **(ITML)**. The regularizer in the objective function in the online version of ITML involves positive definiteness of M using the log-determinant (LogDet) divergence between the optimal M and some intermediate M^i in iteration i, and is defined as

$$D_{ld}(M, M^i) = \operatorname{tr}(M(M^i)^{-1}) - \log\det(M(M^i)^{-1}) - n \tag{3.31}$$

where n is the dimension of the matrix $M, M^i \in \mathbb{R}^{n\times n}$. This term assures positive definiteness of the matrix M^i. Furthermore, it can easily be verified that $r((M^i)^{-1/2}M(M^i)^{-1/2}) = D_{ld}(M, M^i)$. The motivation of using the LogDet divergence as a regularizer is due to its nice beneficial properties during numerical optimization. In fact the LogDet divergence as defined in (3.31) is scale and translation invariant and is only defined for positive definite matrices M^i. Furthermore, when considering the data to be multivariate Gaussian, the distance between two multivariate Gaussian distributions with the same mean μ and covariances M and M^i can be measured with the Kullback-Leibler (KL) divergence, see (2.100) and the following identity holds:

$$\mathrm{KL}(p(x;\mu, M^i)||p(x;\mu, M)) = \frac{1}{2}D_{ld}(M, M^i). \tag{3.32}$$

The loss in ITML at iteration i is taken as the quadratic loss given by

$$l_i = (d^2_M(x_i, x'_i) - d^2_{M^i}(x_i, x'_i))^2 \tag{3.33}$$

which measures the quadratic distance between the true distance and the predicted distance.

Another similar algorithm to ITML was proposed in [JKDG08], ***LogDet Exact Gradient Online*** **(LEGO)** which also involves a LogDet divergence in the regularizer of the objective.

In [GH11] the authors proposed the first approximate solver for semi-definite programs which runs in sublinear time by applying a linear complexity algorithm only on a subset of the data. The proposed algorithm in [GH11] was applied to the problem of metric learning and processes only a fraction of the set of triples $\mathcal{T}$ with a guarantee to get an ε approximate solution.

The authors use the same objective as in (3.28) with $b = 1$, while taking into account that a minimum margin of separation γ should be maximized. Then the objective can be rewritten as a $\max - \min$ problem

$$\max_{M\succeq 0} \min_{i\in[|\mathcal{T}|]} y_i(1 - d^2_M(x_i, x'_i)). \tag{3.34}$$

Additionally M is constrained to be in the bounded cone of positive semi-definite matrices defined by

$$\mathcal{P} = \{M | M \succeq 0 \text{ and } \operatorname{tr}(M) \leq 1\}. \tag{3.35}$$

The resulting optimization problem can be rewritten using simplex constraints

$$\max_{M \in \mathcal{P}} \min_{u_i \in \triangle_{|\mathcal{T}|-1}} \sum_{i=1}^{|\mathcal{T}|} u_i y_i \Big(1 - d_M^2(x_i, x_i')\Big). \tag{3.36a}$$

3.4. Numerical Optimization Techniques

After discussing the most representative models for metric learning we present the optimization techniques used in the literature for solving the optimization problems, some being semi-definite programming solvers specialized to a particular problem.

3.4.1. Gradient Descent and Projected Gradient Descent

From the general formulation of the metric learning problem (3.3) the domain of M is constrained to be the cone of positive semi-definite matrices $\mathbb{S}_+^n$. As a result basic gradient descent method, see Sect. 2.2.3, suitable for (convex) differentiable unconstrained problems will not work. However, when decomposing M as in (3.8) as $M = L^\top L$, which is implied by the positive semi-definiteness of M, the metric learning problem (3.3) can be solved as an *unconstrained* optimization problem in L. The resulting non-convex problem can then be optimized using a gradient descent method, however with no guaranty to find the global optimum. As the models we presented in the previous section do not rely on this factorization, we do not further consider this class of optimization methods in the following.

One approach to solving the convex constrained problem (3.8) is the projected gradient descent method [LP66], that is a gradient descent with a projection step to the cone of positive semi-definite matrices. The gradient projection update step on the general metric learning problem (3.3) can be written as

$$M^{i+1} = \mathrm{P}_{\mathbb{S}_+^n}\Big(M^i - \lambda^i \nabla\big(l(M, \mathcal{S}, \mathcal{D}, \mathcal{R}) + \delta r(M)\big)\Big). \tag{3.37}$$

The projection P on $\mathbb{S}_+^n$ can be implemented by computing the eigenvalues of the resulting matrix after the gradient descent step $M^i - \lambda^i \nabla\big(l(M, \mathcal{S}, \mathcal{D}, \mathcal{R}) + \lambda r(M)\big)$ and equating the negative ones to zero. In order to ensure convergence, the step size λ^i should be chosen appropriately, for instance using line search such that the Wolfe conditions (2.34) are satisfied.

Projected gradient descent was also applied by Xing et al. [XNJR02] for optimizing the constrained problem (3.9). Beside the constraint on the positive semi-definiteness of M, the metric learning problem introduces an additional constraint on the metric for the set of similar points. Due to this the authors propose 2 successive projections on the corresponding sets.

3.4.2. Minimizing the Maximal Eigenvalue of a Symmetric Matrix

Minimizing the maximum eigenvalue of a matrix is a very important class of eigenvalue optimization tasks [LO96] which can be formulated as a convex but non-smooth optimization problem. It has been well studied in the literature and we refer the reader to [Ove88, OW93, Ous00, SF95] for a detailed discussion. A very important result for us is that semi-definite programs with constant trace on the primal feasible set can be equivalently reformulated as maximum eigenvalue optimization problems. Furthermore, the maximum eigenvalue function on a symmetric matrix as defined in (3.17) is differentiable only when the multiplicity of the maximum eigenvalue is 1 [OW93]. However, in most cases the optimal matrix has a maximum eigenvalue with a multiplicity larger than 1.

Ying et al. in [YL12] first propose reformulating the metric learning problem as minimizing the maximum eigenvalue of a symmetric matrix. The metric learning problem was formulated as a composite function of the maximum eigenvalue function and a linear function (3.16). The authors propose to first approximate the original function

$$f(x) = \max\{x_1, ..., x_n\}, \tag{3.38}$$

by a smooth function

$$f_\mu(x) := \mu \ln \Big(\sum_{i=1}^{n} \exp\big(\frac{x_i}{\mu}\big) \Big). \tag{3.39}$$

Ying et al. propose a new method to optimize the smoothed problem. The authors apply the Frank-Wolfe algorithm [FW56] and its extension to semi-definite programs over the cone of positive semi-definite matrices restricted to trace 1, proposed by Hazan [Haz08] and a smoothing technique of Nesterov [Nes05].

Smooth approximation of the maximum eigenvalue function optimized with a globally convergent method was additionally proposed in [CQQT04].

A bundle method for optimizing the maximum eigenvalue function was developed in [HR00], called a spectral bundle method as it uses more information on the spectrum of the matrix. The authors propose a specific variant of the cutting plane model in their bundle algorithm. Subsequently, they reformulate a semi-definite program without a constraint of the matrix trace to a maximum eigenvalue problem. First, a dual eigenvalue function of the semi-definite program is formulated and then an approximating minorant of the dual function is optimized using a proximal point optimization approach.

In [DK14] a stochastic smoothing algorithm was proposed for minimizing the maximum eigenvalue of a matrix on a convex set. The idea of the algorithm is based on perturbations. Gaussian smoothing or perturbing the argument of the function with a scaled outer product of a random vector from Gaussian distribution with rank one leads to increasing the spectral gap, i.e. the difference between the largest and the second largest eigenvalue. As a consequence, the perturbed metric has a unique maximum eigenvalue. Due to this property, this smoothing always leads to a maximum eigenvalue function which is always differentiable.

3.4.3. Stochastic Gradient

A ***stochastic gradient descent*** **(SGD)** [Bot98] method is a gradient descent method where the gradient of an objective represented by a sum of functions, is approximated by the gradient of *one* function only. Due to this, stochastic gradient is an appropriate method for optimization in online metric learning where at each iteration one loss function and one triple of the data for the particular iterate is provided. As a consequence, SGD allows very fast and memory efficient updates. In an online metric learning scenario with loss function l_i in iteration i depending on the current matrix M^i, the SGD update is given by

$$M^{i+1} = M^i - \lambda^i \nabla l_i(M^i) \tag{3.40}$$

where λ^i is an appropriately chosen step size.

We saw that in the general case for online metric learning at every iterate the metric is updated together with a margin value of separation. While this margin value of separation is maximized, the loss should be minimized. Additionally in online programming with SGD the notion of orthogonal projections of matrices on a convex closed set C is considered, i.e.

$$\mathrm{P}_C(M) = \arg\min_{M^i \in C} \frac{1}{2}||M - M^i||_F^2 \tag{3.41}$$

such that the new projected matrix $M^i \in C$ is the one closest to M. So the updated matrix from the online algorithm should be as close as possible to the previous matrix and should minimize the objective (3.29) on the other hand.

For instance the online metric learning version of the algorithm described in [SSSN04] is performed using two projection steps: one on the cone of positive semi-definite matrices, and another one projecting γ on $\max\{1, \gamma\}$. It turns out that due to the eigenvalue interlacing theorem [GVL96] the rank one update of the matrix M causes only at most one eigenvalue to be negative.

In ITML [DKJ+07] and LEGO [JKDG08] the orthogonal projection update as in (3.41) is generalized to the LogDet divergence (3.31) and this is called mirror descent [BT03] method, a generalization of the SGD. No projection to the cone of positive semi-definite matrices is required as the LogDet divergence guarantees this.

For the sublinear online learning in [GH11] the objective is optimized using an approximate variant of the SGD, i.e. the projections on the bounded positive semi-definite cone (3.35) are computed only approximately. The loss is updated using the multiplicative weights method [AHK12].

The online metric learning algorithm in [GH11] can be thought of as some variant of a bundle method, algorithm for optimizing convex non smooth functions. Bundle methods use piecewise affine approximation of the objective function which is iteratively refined, such that the final piecewise affine function is a good approximation of the function in the optimal point. The gradient step is performed on this approximating *function*, whereas in online learning we have an approximate *gradient* to the original function which updates the metric.

Performance of online learning algorithms, is normally measured with a regret value defined after a certain number of T iterations of the algorithm, with

$$R_T = \sum_{i=1}^{T} l_i(M^i) - \sum_{i=1}^{T} l_i(M^*) \tag{3.42}$$

where l_i is the loss at some iteration i as a function of the metric matrix M^i and the optimal metric M^* respectively. The regret measures the difference between the sum of all losses after certain number of iterations T of the algorithm and the sum of optimal losses, $l_i(M^*)$. The regret can also be considered as a measure of goodness of the best possible solution the online algorithm returns when compared to a solution of a batch algorithm which processes all data at one time.

For more on the topic on online convex programming we refer to [CBL06, Zin03].

3.5. Proposed Approach

In this work we propose two new optimization approaches for the metric learning problem formulated by Xing et al. [XNJR02], see (3.9) and the metric learning problem when formulated as an eigenvalue optimization problem by Ying et al. [YL12], see (3.16). Our main idea is to use interior point methods to lift the constraints from these two metric learning problems into the objective function. In the following sections we first prove some basic results on the two above mentioned metric learning problems and present our optimization approach. We use both a first order method and a second order Newton method for the inner iterations of the barrier method. We demonstrate the efficiency of our optimization technique with numerical experiments while comparing it to Matlab software for disciplined convex programming (CVX) [GB].

3.5.1. Objective Functions

In Sect. 3.3.1 we introduced one of the earliest and well known approaches in Mahalanobis distance learning. The work of [XNJR02] proposes to maximize distance between dissimilar points while the squared distance between similar points is upper bounded, under the constraint $M \in \mathbb{S}_+^n$. We reformulate here again the problem from (3.9) as a minimization problem:

$$\min_{M \succeq 0} \sum_{(i,j)\in\mathcal{D}} -\sqrt{\operatorname{tr}(MX_{ij})} \tag{3.43a}$$

$$\text{s.t. } \operatorname{tr}(MX_{\mathcal{S}}) \leq 1 \tag{3.43b}$$

with X_{ij} and $X_{\mathcal{S}}$ as defined in (3.12).

We first prove the convexity of the function in (3.43) under the constraint of positive semi-definiteness of M. Also note that the constraint (3.43b) is linear and convex.

Proposition 3.5.1. *The following problem is a convex optimization problem*

$$\min_{M \succeq 0} \sum_{(i,j)\in\mathcal{D}} -\sqrt{\mathrm{tr}(MX_{ij})} \tag{3.44}$$

with X_{ij} as defined in (3.12).

Proof. We require convexity only on the feasible set $\mathbb{S}^n_+$ and in the following when taking about the objective function in (3.44) we consider its restriction on the feasible set. We can prove the convexity of the objective function in (3.44) using the Hessian. A twice differentiable function $f(x)$ is convex if and only if its Hessian is a positive semi-definite matrix, $\nabla^2 f(x) \succeq 0$. We have

$$f(M) = -\sum_{(i,j)\in\mathcal{D}} \sqrt{\mathrm{tr}(MX_{ij})}. \tag{3.45}$$

A sum of convex functions is a convex function so it is enough to prove the convexity of one of the functions inside the sum in (3.45). We have to prove convexity of

$$f_{ij}(M) = -\sqrt{\mathrm{tr}(MX_{ij})} \text{ where } (i,j) \in \mathcal{D}. \tag{3.46}$$

The square root is well defined because $tr(MX_{ij}) = \langle M^\top, X_{ij}\rangle = \langle M, X_{ij}\rangle$ is an inner product of two positive semi-definite matrices and thus is non-negative due to the self-duality of the positive semi-definite cone. For the gradient and Hessian of f_{ij} respectively we have

$$\nabla f_{ij}(M) = -\frac{X_{ij}^\top}{2\sqrt{\mathrm{tr}(MX_{ij})}} \tag{3.47a}$$

$$\nabla^2 f_{ij}(M) = \frac{X_{ij}^\top \otimes X_{ij}^\top}{4(\mathrm{tr}(MX_{ij}))^{3/2}} \tag{3.47b}$$

where by $\otimes$ we denote the Kronecker product. The matrix X_{ij} is a positive semi-definite matrix as an outer product of vectors, which implies that all the eigenvalues of X_{ij} are non-negative. Using the property that the eigenvalues of a Kronecker product of matrices $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times m}$, with eigenvalues λ_i, $i = 1, .., n$ and μ_j, $j = 1, ..., m$, respectively, are $\lambda_i\mu_j$, it follows that as a product of non-negative values they are all non-negative as well. So finally all the Hessians are positive semi-definite, $\nabla^2 f_{ij}(M) \succeq 0$, from which follows that $f(M)$ is convex and thus (3.44) is a convex optimization problem. □

The second objective on which we want to apply a different optimization technique using interior point methods is the equivalent eigenvalue optimization problem to the metric learning problem formulated by Ying et al. [YL12], see (3.16). Inspired by the approach in [XNJR02], the authors in [YL12], instead of maximizing the sum of all the distances between pairs in the dissimilarity set, propose a bit different approach. They maximize the minimal squared distance for pairs with indices in $\mathcal{D}$,

under the constraint that the squared distance sum for pairs with indices in $\mathcal{S}$ is upper bounded, as well as the positive semi-definiteness of M. Another difference in this reformulation mainly is that now instead of taking the Mahalanobis distance which includes a square root, the squared Mahalanobis distance is used, which leads to a linear function. This was formulated as in (3.11). This crucial difference enables a different reformulation as an eigenvalue optimization problem (3.16).

As optimizing the maximum eigenvalue of a symmetric matrix is in general a non-smooth convex problem, the authors in [YL12] propose smoothing the eigenvalue function with a log-exponential function (3.39). In order to derive this formulation, we start from problem (3.22a) with the trace constraint as in (3.22b). In order to remove the constraint $\langle X_{\mathcal{S}}, M\rangle = 1$ we assume that $X_{\mathcal{S}}$ is invertible (in practice this can be done with adding some small ε term to the diagonal elements of $X_{\mathcal{S}}$), we define

$$\tilde{M} := X_{\mathcal{S}}^{1/2} M X_{\mathcal{S}}^{1/2} \tag{3.48a}$$

$$\tilde{X}_{ij} := X_{\mathcal{S}}^{-1/2} X_{ij} X_{\mathcal{S}}^{-1/2} \tag{3.48b}$$

with $X_{\mathcal{S}}$ and X_{ij} as defined in (3.12). Then $\operatorname{tr}(\tilde{M}) = 1$ if and only if $\langle X_{\mathcal{S}}, M\rangle = 1$. Now we can rewrite the metric learning problem as an equivalent minimization problem with simplex constraints using the above defined matrices,

$$\min_{\tilde{M}\in\mathcal{C}} \min_{u\in\Delta_{|\mathcal{D}|-1}} -\sum_{(i,j)\in\mathcal{D}} u_{ij}\langle \tilde{X}_{ij}, \tilde{M}\rangle \tag{3.49}$$

where

$$\mathcal{C} = \{\tilde{M} \succeq 0 : \operatorname{tr}(\tilde{M}) = 1\}. \tag{3.50}$$

The function in (3.49) can be smoothed by adding an additional negative entropy term $\mu \sum_{(i,j)\in\mathcal{D}} u_{ij} \ln(u_{ij})$, where $\mu > 0$ and μ is very small, e.g. $\mu \approx 10^{-5}$. The new smoothed function is clearly again convex being a sum of convex functions, and is given by

$$\min_{\tilde{M}\in\mathcal{C}} f_\mu(\tilde{M}) = \min_{\tilde{M}\in\mathcal{C}} \left(\min_{u\in\Delta_{|\mathcal{D}|-1}} \underbrace{\Big(-\sum_{(i,j)\in\mathcal{D}} u_{ij}\langle \tilde{X}_{ij}, \tilde{M}\rangle + \mu \sum_{(i,j)\in\mathcal{D}} u_{ij}\ln(u_{ij}) \Big)}_{:=g(u)} \right). \tag{3.51}$$

We further want to eliminate the simplex constraint. For this purpose we incorporate the constraint into the function $g(u)$ as defined in (3.51) by introducing a Lagrange multiplier λ, and take the gradient w.r.t. u:

$$g(u,\lambda) = \sum_{(i,j)\in\mathcal{D}} u_{ij}\langle \tilde{X}_{ij}, \tilde{M}\rangle + \mu \sum_{(i,j)\in\mathcal{D}} u_{ij}\ln(u_{ij}) + \lambda\Big(\sum_{(i,j)\in\mathcal{D}} u_{ij} - 1\Big) \tag{3.52a}$$

$$\nabla_u g(u,\lambda)_{ij} = \langle \tilde{X}_{ij}, \tilde{M}\rangle + \mu(\ln(u_{ij}) + 1) + \lambda \cdot |\mathcal{D}| \tag{3.52b}$$

From the first order optimality conditions (Fermat condition), $\nabla_u g(u,\lambda) = 0$, we can

express u for one pair $(i,j) \in \mathcal{D}$

$$u_{ij} = \exp\Big(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle - 1 - \frac{\lambda}{\mu}\Big) \tag{3.53}$$

and inserting this into the simplex constraints

$$\sum_{(i,j)\in\mathcal{D}} u_{ij} = \sum_{(i,j)\in\mathcal{D}} \exp\Big(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle - 1 - \frac{\lambda}{\mu}\Big) = 1 \tag{3.54}$$

we can solve for λ yielding

$$\lambda = \mu \ln\Big(\sum_{(i,j)\in\mathcal{D}} \exp(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle - 1)\Big). \tag{3.55}$$

Then we can rewrite u as

$$u_{ij} = \frac{\exp\Big(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\Big)}{\sum_{\theta\in\mathcal{D}} \exp\Big(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\Big)} \tag{3.56}$$

where we used a different notation θ for the pairs with indices in $\mathcal{D}$ in the denominator

for clearness. Inserting the result into $g(u)$ as defined in (3.51), we get

$$g(u) = \sum_{(i,j)\in\mathcal{D}} \frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)} \langle \tilde{X}_{ij}, \tilde{M}\rangle \tag{3.57a}$$

$$+ \mu \sum_{(i,j)\in\mathcal{D}} \frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)} \ln\left(\frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)}\right) \tag{3.57b}$$

$$= \sum_{(i,j)\in\mathcal{D}} \frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)} \langle \tilde{X}_{ij}, \tilde{M}\rangle \tag{3.57c}$$

$$+ \mu \sum_{(i,j)\in\mathcal{D}} \frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)} \tag{3.57d}$$

$$\cdot \left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle - \ln\left(\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)\right)\right) \tag{3.57e}$$

$$= -\mu \sum_{(i,j)\in\mathcal{D}} \frac{\exp\left(-\frac{1}{\mu}\langle \tilde{X}_{ij}, \tilde{M}\rangle\right)}{\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)} \ln\left(\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)\right) \tag{3.57f}$$

$$= -\mu \sum_{(i,j)\in\mathcal{D}} u_{ij} \ln\left(\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)\right) \tag{3.57g}$$

$$= -\mu \sum_{(i,j)\in\mathcal{D}} \ln\left(\sum_{\theta\in\mathcal{D}} \exp\left(-\frac{1}{\mu}\langle \tilde{X}_{\theta}, \tilde{M}\rangle\right)\right) \tag{3.57h}$$

where in the last equation we used that $\sum_{(i,j)\in\mathcal{D}} u_{ij} = 1$. Finally the new smoothed problem without a simplex constraint is given by

$$\min_{\tilde{M}\in\mathcal{C}} f_{\mu}(\tilde{M}) = \min_{\tilde{M}\in\mathcal{C}} \mu \log\left(\sum_{(i,j)\in\mathcal{D}} \exp(-\langle \tilde{X}_{ij}, \tilde{M}\rangle/\mu)\right) \tag{3.58}$$

with the spectrahedron $\mathcal{C}$ as defined in (3.50).

Now after having derived the final objectives in the next section we propose a new optimization technique for the metric learning problem with objectives as defined in (3.43) and (3.58).

3.5.2. Optimization

In the following we propose an optimization approach for the metric learning problems (3.43) and (3.58) which significantly differ from previous work.

Xing et al. in [XNJR02] solve the optimization problem (3.43) with the projected gradient method, see Sect. 3.4.1. Ying et al. in [YL12] solve (3.58) with an approximate Frank-Wolfe algorithm, appropriately designed for their particular function.

In contrast, we propose a simpler first order gradient descent method as well as a second order (damped) Newton method. As both require unconstrained objectives, we first introduce log-barrier functions to approximate the indicator functions of the constraint sets. Before proposing our method we give a small introduction on interior point methods, part of which are barrier methods.

Interior Point Methods

The main idea with interior point methods as the name suggests is to force all iterates of the algorithm to stay in the interior of the feasible set. To this end, barrier functions are introduced which are only defined on the inside of the feasible set. For instance, logarithmic (log) barrier functions approximating a non-negativity constraint are only defined on the positive real line.

Interior point methods date back to Karmarkar [Kar84] and we refer the reader to [Ber99, BV04, Tod01, NT08, PW00] for more details.

Let us consider a constrained optimization problem

$$\min f(x) \tag{3.59a}$$
$$\text{s.t. } g_i(x) \geq 0 \qquad i = 1, ..m \tag{3.59b}$$

with $f, g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, .., m$ convex, continuous differentiable and real valued functions. We can lift the constraints into the objective function by introducing an indicator function $I : \mathbb{R} \to \mathbb{R}$, yielding

$$\min f(x) - (\sum_{i=1}^{m} I(g_i(x)) \tag{3.60}$$

and I being defined by

$$I(x) = \begin{cases} 0 & \text{if } x \geq 0 \\ \infty & \text{if } x < 0. \end{cases} \tag{3.61}$$

The idea of barrier methods is to approximate this indicator function I. Two most common approximations of the indicator function as defined above, are the ***log barrier*** function given by

$$\hat{I}(x) = -\delta \log(x) \tag{3.62}$$

where $\text{dom}(\hat{I}) = \mathbb{R}_+$, the set of positive real numbers, and the ***inverse*** function

$$\hat{I}(x) = -\delta \frac{1}{x} \tag{3.63}$$

where $\text{dom}(\hat{I}) = \mathbb{R} \setminus \{0\}$ and δ is a parameter used to set the accuracy of the approximation. In the following we use the log barrier function to convert the constrained problem in (3.59) in an unconstrained one:

$$\min f(x) - \sum_{i=1}^{m} \delta_i \log(g_i(x)). \tag{3.64}$$

The added log barrier function goes to ∞ as $g_i(x) \to 0$.

As δ_i gets smaller the unconstrained problem in (3.64) is a better approximation to the original function in (3.59). In practice it has shown to be beneficial to choose δ_i as a continuously decreasing sequence $\{\delta_{i,k}\}_k$ of each of δ_i such that $\lim_{k\to\infty} \delta_{i,k} = 0$, for $i = 1, .., m$. It can be shown that as $x_k \to x_*$, $\delta_{i,k} \log(g_i(x)) \to 0$ for $i = 1, .., m$. This was stated and proved in Proposition 4.1.1 in [Ber99].

Proposition 3.5.2. *Every limit point of a sequence $\{x_k\}_k$ generated by a barrier method, where the parameters δ_i are chosen as a continuous decreasing sequences converging to zero, is a global minimum of the original constrained problem* (3.59).

The function in (3.64) is a convex differentiable function and can be optimized with first order gradient descent method. When f is twice differentiable and convex second order Newton method can be applied as well.

Optimization with Barrier Method

In a first step we consider the constrained optimization problems (3.43) and (3.58). We replace the constraints by introducing log barrier functions to the objective to gain an unconstrained version of the problem.

We denote the new objective function by f_{Xing} and the overall unconstrained version of (3.43) then reads:

$$\min_M f_{\text{Xing}}(M) = \min_M - \sum_{(i,j)\in\mathcal{D}} \sqrt{\text{tr}(MX_{ij})} - \delta_1 \log(1 - \text{tr}(MX_{\mathcal{S}})) - \delta_2 \log(\det(M)) \tag{3.65}$$

where X_{ij} and $X_{\mathcal{S}}$ are as defined in (3.12). δ_1 and δ_2 are parameters which have to be properly chosen, in order to obtain a sequence of iterates that converge to a global optimal solution, see Proposition 3.5.2. We will get back on how to choose exactly δ_1 and δ_2 when we describe our algorithm we use. Since our function $f_{\text{Xing}}(M)$ as formulated above is convex and differentiable we can apply a gradient descent method. For the gradient of f_{Xing} we have

$$\nabla f_{\text{Xing}}(M) = - \sum_{(i,j)\in\mathcal{D}} \frac{X_{ij}^\top}{2\sqrt{\text{tr}(MX_{ij})}} + \delta_1 \frac{X_{\mathcal{S}}^\top}{1 - \text{tr}(MX_S)} - \delta_2 (M^{-1})^{-\top}. \tag{3.66}$$

Remark 3.5.1. Concerning the last term, the inverse of the positive semi-definite matrix M, we note that it can happen that M has zero eigenvalues in which case we have a singular matrix. In order to exclude this we aways add a diagonal matrix, for instance $I \cdot \varepsilon = I \cdot 10^{-6}$ to M, before inverting it.

As for the second objective formulation from (3.58) we have one inequality constraint, $\tilde{M} \succeq 0$ and one equality constraint, $\text{tr}(\tilde{M}) = 1$. For the inequality constraint we use again a log barrier function, while for the equality constraint we use the method of Lagrange multipliers see Sect. 5.1.1 in [Ber99]. The final objective depending on

two variables is given by

$$\min_{\tilde{M}} \max_{\lambda} L(\tilde{M}, \lambda) = \min_{\tilde{M}} \max_{\lambda} \mu \log\Big(\sum_{(i,j)\in\mathcal{D}} \exp\Big(- \langle \tilde{X}_{ij}, \tilde{M}\rangle/\mu\Big)\Big) - \delta_1 \log(\det(\tilde{M})) + \lambda(1 - \mathrm{tr}(\tilde{M})) \tag{3.67}$$

with $\tilde{X}_{ij}$ and $\tilde{M}$ as defined in (3.48), and appropriate δ_1. We optimize this objective by alternating between the maximization of λ and the minimization of $\tilde{M}$. The function is clearly convex and differentiable and for the gradient with respect to $\tilde{M}$ we have

$$\nabla_{\tilde{M}} L(\tilde{M}, \lambda) = -\frac{\sum_{(i,j)\in\mathcal{D}} \exp\Big(- \langle \tilde{X}_{ij}, \tilde{M}\rangle/\mu\Big) \cdot \tilde{X}_{ij}}{\sum_{(i,j)\in\mathcal{D}} \exp\Big(- \langle \tilde{X}_{ij}, \tilde{M}\rangle/\mu\Big)} - \delta_1 (\tilde{M}^{-1})^\top - \lambda I. \tag{3.68}$$

For the gradient with respect to λ we have $\nabla_\lambda L(\tilde{M}, \lambda) = 1 - \mathrm{tr}(\tilde{M})$. We point out that even though we use method of Lagrange mutlipliers to handle the linear equality constraints, in combination with barrier method, this is still considered as an "extension" of the barrier method as Boyd et al. call it in [BV04].

We next describe the barrier method we use for optimizing (3.65) and (3.67) in combination with gradient descent and Newton method.

Barrier Method with Gradient Descent and Newton Method

Algorithm 1: Barrier Method with Gradient Descent

input : feasible matrix M^0, so that the constraints of the original problem are satisfied, number of inequality constraints m, tolerance value $\varepsilon > 0$, $\alpha > 0$, $\delta_{i,k} = \delta_{i,k_0}$, $i = 1, ..m$, k outer iteration number, I inner iteration number

```
while δ_{i,k} · m > ε do
    while ||∇f(M^I)||_2 > ε do
        compute direction of the gradient descent ∇f(M^I) ;
        compute step size t with line search so that Wolfe conditions (2.34) are
         satisfied ;
        M^I = M^I + t∇f(M^I);
        I = I + 1;
    δ_{i,k} = δ_{i,k}/α  i = 1, ..m
     return M^I
```

We have two (unconstrained) convex differentiable problems in (3.65) and (3.67). One way to globally optimize them is with steepest descent method. As for the second objective (3.67) where we have a saddle point problem, we use alternate optimization. We maximize with respect to λ using gradient ascent and minimize with respect to $\tilde{M}$ using a barrier method with gradient descent for the inner iterations. In this case

Algorithm 2: Barrier Method with Newton Method

input : feasible matrix M^0, so that the constraints of the original problem are satisfied, number of inequality constraints m, tolerance value $\varepsilon > 0$, $\alpha > 0$, $\delta_{i,k} = \delta_{i,k_0}$, $i = 1, ..m$, k outer iteration number, I inner iteration number

1 **while** $\delta_{i,k} \cdot m > \varepsilon$ **do**
2 **while** $N^2/2 > \varepsilon$ **do**
3 compute Newton step $S = \nabla^2 f(M^I)^{-1} \nabla f(M^I)$ and Newton decrement $N = (\nabla f(M^I)^\top \nabla^2 f(M^I)^{-1} \nabla f(M^I))^{1/2}$;
4 compute step size t with backtracking line search ;
5 $M^I = M^I + tS$;
6 $I = I + 1$;
7 $\delta_{i,k} = \frac{\delta_{i,k}}{\alpha}$ $i = 1, ..m$
8 **return** M^I

in line (7) in every outer iteration of Algorithm 1 we additionally perform the ascent step for λ, $\lambda = \lambda + s^k(1 - \mathrm{tr}(\tilde{M}))$, with some step size s^k, where k is the current number of the outer iterations. In the experiments here we took for the step size $s^k = \frac{1}{2k}$. In order to not cause confusion we did not include this in the pseudo-code of Algorithm 1. The barrier method uses properly chosen parameters $\delta_{i,k}$, $i = 1, ..., m$ in the outer iteration k and line search for choosing an appropriate step size for the gradient descent in the inner loop.

Our functions in (3.65) and (3.67) are twice differentiable so another possibility is to use the Newton method for the inner iterations. To this end we replace lines (2-6) in Algorithm 1 by a (damped) Newton update step for M^I combined with backtracking line search, see Algorithm 2. For some background of the gradient descent and the Newton method we refer to Sect. 2.2.3 and Sect. 2.2.5.

Note that it is important for the barrier method to be initialized with a feasible point. We point out that concerning the inner iterations it is not always required to solve until convergence as Boyd et al. argue in [BV04]. Instead we use as fixed the number of inner iterations. The parameter α controls the number of outer iterations.

Convergence Analysis of the Barrier Method in Algorithm 1 and Algorithm 2

For the outer iterations or centering steps we have the following theorem, see Sect. 11.3.3 in [BV04].

Theorem 3.5.3. *After k outer iterations (centering steps) of the barrier method we have*

$$f(x^k) - f^* \leq \frac{m\delta_{i,0}}{\alpha^k} \tag{3.69}$$

where f^* is the optimal value of the function we want to achieve, m is the number of constraints in the original function and α is the factor by which the parameters δ_i, with $\delta_{i,0}$ the initial ones, $i = 1, ..., m$ are decreasing.

As long as the inner iterations with the gradient descent are solvable, the barrier

method converges to the optimal solution. What remains is the speed of convergence of gradient descent for different choices of values for the decreasing sequences of δ_i. Gradient descent can be sometimes very slow but on the other hand is very simple. In order to converge, gradient descent requires the sublevel set $S_-(f, f(x^0))$ to be closed, see Definition 2.2.15. This condition is always satisfied for closed functions. A function f is closed if f is continuous and the dom f is closed, or if dom f is open and $f(x) \to \infty$ as x approaches the boundary of dom f. In our case the second condition for a closed function is satisfied due to the log barrier functions. Under the assumption that the sublevel set $S_-(f, f(x^0))$ is closed, and the function f is strongly convex, see Definition 2.2.10, convergence of gradient descent can be analyzed using the condition number $\frac{m}{M}$ of $S_-(f, f(x^0))$ where $mI \leq \nabla^2 f \leq MI$. In order to see if we can say something about the condition number we first need to compute the Hessians of the two functions (3.65) and (3.67).

The Hessian of (3.65) is given by

$$\nabla^2 f_{\text{Xing}}(M) = \frac{X_{ij}^\top \bigotimes X_{ij}^\top}{4(\text{tr}(MX_{ij}))^{3/2}} - \delta_1 \frac{X_{\mathcal{S}} \otimes X_{\mathcal{S}}}{(1 - \text{tr}(MX_{\mathcal{S}}))^2} + \delta_2(\tilde{M}^{-\top} \otimes \tilde{M}^{-1}). \tag{3.70}$$

Let us denote

$$s_{ij}^{\tilde{M}} := \exp\Big(- \left\langle \tilde{X}_{ij}, \tilde{M} \right\rangle / \mu \Big). \tag{3.71}$$

Then the Hessian of (3.67) with respect to $\tilde{M}$ is given by

$$\begin{aligned} \nabla^2_{\tilde{M}} L(\tilde{M}, \lambda) = & \underbrace{- \frac{\Big(\sum_{(i,j)\in\mathcal{D}} s_{ij}^{\tilde{M}} \cdot \tilde{X}_{ij} \Big) \otimes \Big(\sum_{(i,j)\in\mathcal{D}} s_{ij}^{\tilde{M}} \cdot \tilde{X}_{ij} \Big)}{\mu \Big(\sum_{(i,j)\in\mathcal{D}} s_{ij}^{\tilde{M}} \Big)^2}}_{I_1} \\ & + \underbrace{\frac{\sum_{(i,j)\in\mathcal{D}} s_{ij}^{\tilde{M}} \cdot \tilde{X}_{ij} \otimes \tilde{X}_{ij}}{\mu \sum_{(i,j)\in\mathcal{D}} s_{ij}^{\tilde{M}}}}_{I_2} + \underbrace{\delta_1(\tilde{M}^{-\top} \otimes \tilde{M}^{-1})}_{I_3}. \end{aligned} \tag{3.72}$$

The Hessian (3.70) does not allow to conclude much on the boundedness, except that $\nabla^2 f(M) \geq 0$, as we already proved in Proposition 3.5.1. For the Hessian (3.72) of the second objective, we can find upper bounds for terms I_1 and I_2, see [YL12, Lemma 4], but none can be found for I_3.

If we can prove that both of the objective functions are self-concordant, see Definition 2.2.16 we can make use of this property to conclude on the convergence of the Newton method [BV04]. While the log barrier functions are self-concordant we do not know about the main part of the objective for (3.65) and (3.67).

3.5.3. Experiments and Discussion

In this section we empirically demonstrate the efficiency of the introduced optimization techniques and in general the objective of metric learning approaches.

To this end we investigate the following aspects:

1. We compare k-means clustering based on Euclidean distance similarity measure to k-means clustering based on metric learned distance, in order to demonstrate the efficiency of a learned distance.

2. We compare our optimization technique to established semi-definite solvers in order to demonstrate the ability to find a solution to the optimization problems.

We compare our proposed optimization approach applied to the unconstrained problems (3.65) and (3.67) to specialized semi-definite solvers applied to the original constrained optimization problems (3.43) and (3.58) as proposed in the corresponding papers [XNJR02] and [YL12], respectively. For implementation we use the disciplined convex programming (CVX, see [GB]) framework in Matlab.

Notation and Technical Details We first introduce the technical details and notation for the combination of the optimization techniques on the objective functions which we implement.

When implementing the barrier method with gradient descent for the inner iterations as in Algorithm 1 applied on the objective (3.65) we denote it with **Xing (Grad)**, while when implementing the barrier method with Newton method for the inner iterations as in Algorithm 2 we denote it with **Xing (Newt)**. When implementing Algorithm 1 on the objective (3.67) we denote it with **Ying (Grad)**. When the objective (3.43) from Xing et al. [XNJR02] is optimized with a semi-definite solver from CVX [GB], Sedumi we denote it with **Xing (CVX)**, while when optimizing the objective (3.58) from Ying et al. [YL12] with Sedumi from CVX [GB] we denote it with **Ying (CVX)**.

After learning a similarity distance measure in order to illustrate the mapped data to a different space after learning, we multiply the original data with the lower triangular matrix L from the Cholesky decomposition [GVL96] of $M = LL^\top$.

For the reported results we use an error measure of the total number of misclassified pixels defined by

$$\text{Error} := 100\frac{\sum_{i=1}^{N}(l(i) - l_{gt}(i))}{N}, \tag{3.73}$$

where N is the total number of instances, l is the obtained label (class) and l_{gt} the ground truth label.

We normalize the features from the datasets we use to be in the range of $[0, 1]$ in order all the features to be taken into account equally. This affects the result from k-means clustering as well.

Toy Data Experiment As data we generate small sets with two classes with 30 instances for each class, Fig. 3.1(a). As features we use the (x, y) position of the data.

The distance between the data points in each of the two clusters varies and some data points are spread further from the cluster center. The two clusters also have some points on the "borders". This leads to conclusion that k-means clustering [Llo82] will fail to cluster some of the data appropriately. This is illustrated in

Fig. 3.1(b) where the 4 green points denote incorrectly clustered data which amounts to 6.67% of the data.

After learning the similarity metric distance with Xing (Grad) the data is mapped as in Fig. 3.1(c) and clustered with k-means applied to the new distance as in Fig. 3.1(d). The new metric tends to project the data such that data within each cluster is brought as close as possible while the clusters tend to get sparated by some margin. As a result this leads to perfect clustering of the data. The same results are obtained when learning with Xing (Newt) and Xing (CVX). The results after learning with Xing (CVX) are illustrated in Fig. 3.1(e) and Fig. 3.1(f).

As for learning with Ying (Grad) the new mapped data after learning is illustrated in Fig. 3.2(a) and in Fig. 3.2(b) after clustering. The projection of the data is worse than in the case for Xing(Grad), Xing(Newt) and Xing(CVX) but still slightly better than the original data. As a result after k-means clustering only one point gets wrongly clustered. When applying Ying (CVX) the solver fails to return an optimal solution. The reason can be due to the logarithm and exponential function in the objective which CVX approximates using Taylor expansions.

The parameters from Algorithm 1 and Algorithm 2 were chosen as following: $m = 2$ for Xing (Grad) and Xing (Newt) and $m = 1$ for Ying (Grad). We choose the initial parameters to be $\delta_{i,0} = 1, i = 1, ..., m$. The trade off between the inner and outer iterations was set to $\alpha = 2$, and we used a fixed number of 50 inner iterations. For the tolerance value we choose $\varepsilon = 10^{-3}$ and we initialized with the same feasible matrix $M_{in} = 10^{-1}I/(\mathrm{tr}(X_{\mathcal{S}}I) + 10)$. In order to gain a feasible M as initialization of Ying (Grad) we choose $M_{in} = I/(\mathrm{tr}(X_{\mathcal{S}}I))$ which fulfills the equality constraint $\mathrm{tr}(MX_{\mathcal{S}}) = 1$. As for the k-means implementation we use the default values for Matlab.

Experiment on the Fisher Iris Dataset [Fis36] We perform the same experiments on the Fisher iris dataset [Fis36] which consists of 150 instances, grouped into 3 classes and each instance is given 4 features. We first divide the dataset randomly into 70% training and 30% testing sets. After learning a suitable metric on the training data we perform k-means clustering with the new similarity distance learned on the test dataset. We use the same parameters as for the first experiment on our synthetic data, except for the tolerance value which we set to $\varepsilon = 10^{-5}$. The resulting errors are provided in Table 3.1, including the measurements for the training dataset. Again metric learning significantly improves the classification results. Even tough the resulting metric from Xing (CVX) leads to best clustering performance, still our optimization methods Xing (Grad), Xing (Newt) and Ying (Grad) significantly improve the clustering after learning. Seems that Ying (Grad) learns a better metric on the train data than Xing (Grad) and Xing (Newt) but this metric does not generalize good enough on the test data. The reason that Xing (CVX) is better than our optimization methods can be due to the approximative nature of the barrier method and the parameter choice we use. We have to remark that it is not always clear how to choose α for Algorithm 1 and Algorithm 2. Concerning Ying (Grad) the authors of the corresponding paper propose a sophisticated Frank-Wolfe algorithm

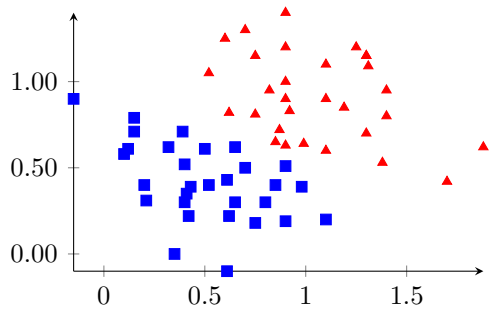
for this objective which is an approximation to the original metric learning method due to smoothing. This suggests that a simple barrier method might not generalize from small data sets to larger ones.

Experiment on the Wine Dataset [Lic13] In addition we perform the same experiments on the wine dataset from the UCI repository [Lic13]. The dataset consist of 178 instances with 13 features each, assigned to 3 classes. We first divide the dataset into 70% for training and 30% for testing. The parameters are chosen the same as in the first experiment, except for $\varepsilon = 10^{-8}$. With Xing (Grad) we learn the metric that clusters the train data leading to the smallest error. With Ying (Grad) and Xing (CVX) the classification result on the train data is the same as before learning. The learned metric from Xing (Grad) and Ying (Grad) does not generalize well to unseen data, the test data in this case and does not lead to improvement after learning. But with Xing (CVX) even tough the train data error stays the same as before learning the test error is improved. Due to high feature dimensionality, we did not use Xing (Newt) for the wine dataset.

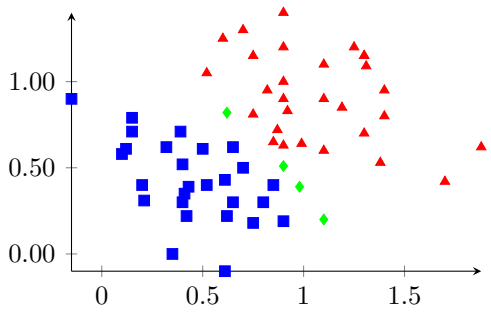
Conclusion With the first toy experiment we illustrated what exactly metric learning tries to achieve. Project data to a space so that similar data is brought as close as possible while dissimilar data is separated as much as possible. As a result after learning a suitable metric the performance of k-means clustering is sufficiently improved. All the experiments account for that. We showed empirically that our proposed optimization techniques Xing (Grad), Xing (Newt) and Ying (Grad) are in most of the cases producing the same or close result as Xing (CVX). This showed the competitiveness of our optimization method to established metric learning solvers.

method	k-means	Xing (Grad)	Xing (Newt)	Xing (CVX)	Ying (Grad)	Ying (CVX)
toy data	6.67	**0**	**0**	**0**	1.67	failed
iris train	5.71	4.76	4.76	**0.95**	3.81	failed
iris test	6.67	4.44	4.44	**2.22**	6.67	failed
wine train	4.03	**2.42**	-	4.03	4.03	failed
wine test	14.81	14.81	-	**11.11**	14.81	failed

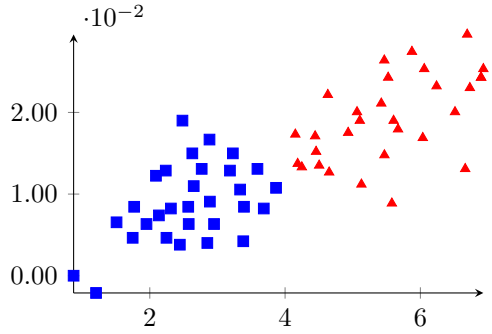
Table 3.1. - Classification errors in percentage on the toy data we generate, the iris dataset [Fis36] and the wine dataset [Lic13]. Errors are compared when the data is clustered using k-means with Euclidean distance measure before learning the similarity metric and k-means after the new distance metric is learned with Xing (Grad) and Xing (Newt). This result is compared with Xing (CVX). These results are further compared with Ying (Grad) and Ying (CVX). However, the Sedumi solver fails to solve this problem to optimality which can be due to the approximation to the logarithm and exponential function in the objective. We can see the huge improvement of the k-means clustering after learning an appropriate similarity metric. As the dimensionality of the features is higher for the wine dataset and we require expensive inverted Hessian for Xing (Newt) we did not implement it.



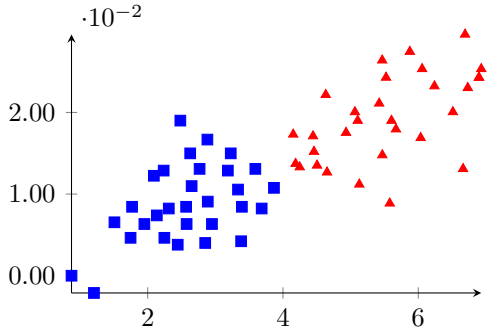
(a) Generated 2D data assigned to two separate clusters



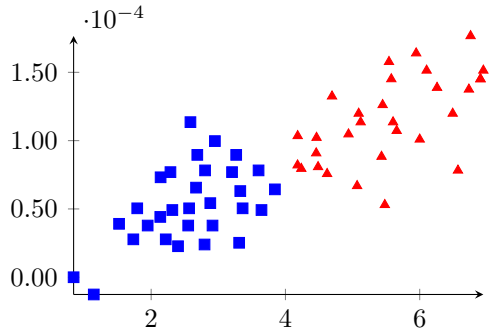
(b) Clustered 2D data from (a) using k-means based on Euclidean distance, green denotes misclassified data



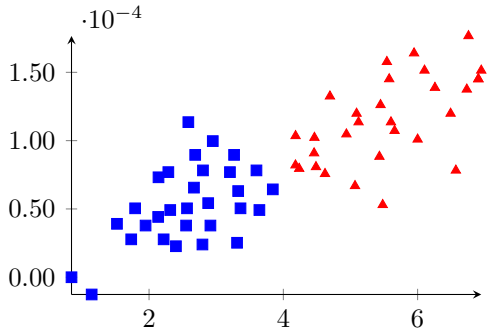
(c) 2D data from (a) mapped to a new space using the learned matrix with Xing (Grad)



(d) Clustered new mapped data from (c) using k-means based on Euclidean distance (k-means using new distance measure)

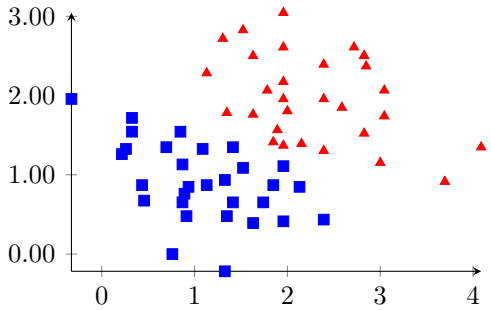


(e) 2D data from (a) mapped to a new space using the new matrix learned with Xing (CVX)

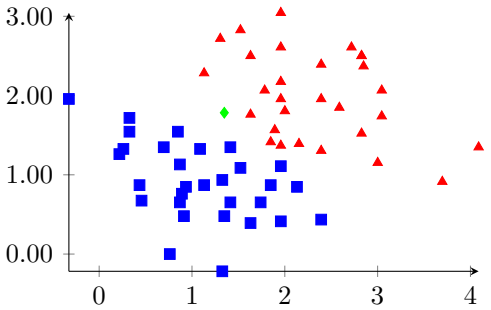


(f) Clustered mapped data from (e) using k-means based on Euclidean distance

Figure 3.1. - (a) Two 2D clusters, (b) clustered with k-means, $k = 2$ leads to 6.67% misclassification. The clustering is ***successful*** when a new distance is used which is learned with Xing (Grad), see (c) for the mapped data using the matrix learned and (d) for the clustering result. Same result is obtained when learning with Xing (CVX), see (e) and (f) for transformed data and clustering result respectively. See text for discussion on the results.



(a) 2D data from Fig. 3.1(a) mapped to a new space using the new matrix learned with Ying (Grad)



(b) Clustered mapped data from (b) using k-means based on Euclidean distance

Figure 3.2. - The original data from Fig. 3.1(a) mapped to a new space using Ying (Grad) (a) and classification result using k-means ***improves*** (b) when compared to Fig. 3.1(b).

4. Model Parameter Perturbation and Learning

4.1. Overview

The learning problem as introduced in Sect. 2.5 is a central element in computer vision, for example in image segmentation [RM03, BDS+09], image denoising [EA06, HKWL14], object recognition [LHB04, Hei03] and scene classification [BLSB04, ZZS14].

The main contribution of this work is two novel learning methods for graphical models. The first one exploits ***inverse linear programming***, see Sect. 2.6, and is to our knowledge the first learning approach to solve the learning problem in graphical models using inverse optimization. The second approach we propose resembles already existing learning methods, like structured Support Vector Machine (SVM) [FJ08, TJHA05, THJA04].

We choose the image labeling problem described in Sect. 2.4.3 as an exemplary scenario to evaluate the proposed learning methods. A labeling can be considered as a mapping from the set of all nodes in a graph to the set of labels in the image. By introducing an energy function measuring the quality of a labeling, the problem consists in finding a labeling that minimizes the energy. Furthermore, minimizing the energy function is equivalent to solving the Maximum a Posteriori (MAP) inference problem which was introduced in Sect. 2.4.2.

Let the discrete energy function be given by

$$E_\theta(x) = \sum_{i \in \mathcal{V}} \theta^i(x_i) + \sum_{ij \in \mathcal{E}} \theta^{ij}(x_i, x_i) \tag{4.1}$$

defined on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the model parameters θ collect the function values of the unary and pairwise potentials θ^i and θ^{ij} respectively. The energy function E_θ evaluates assignments of labels x from a predefined label set $\mathcal{L} = (1, \dots, L)$ to every node $i \in \mathcal{V}$.

Minimizing the energy (4.1) in the general case is NP-hard problem as discussed in Sect. 2.4. A common way to solve it approximately is to relax it to a convex problem, as discussed in Sect. 2.4.1, since convex problems are easier to optimize. MAP estimation in a discrete setting can be written in the form of a linear integer program. Relaxing the integer constraints to an appropriate convex set yields a convex relaxation problem. There are efficient solvers that can find an optimal solution in polynomial time to a relaxed version of (4.1).

We consider the local polytope relaxation [Shl76, Wer07] which enforces consistency

of unary and pairwise variables enclosed in μ via the local polytope $\mathcal{L}_\mathcal{G}$, and with abuse of notation we now write in the minimization problem the energy as $E_\theta(\mu)$ given by

$$\min_{\mu\in\mathcal{L}_\mathcal{G}} E_\theta(\mu) = \min_{\mu\in\mathcal{L}_\mathcal{G}} \sum_{i\in\mathcal{V}} \theta^i(x_i)\mu_i(x_i) + \sum_{ij\in\mathcal{E}} \theta^{ij}(x_{ij})\mu_{ij}(x_{ij}) \tag{4.2}$$

and $\mathcal{L}_\mathcal{G}$ is as defined in (2.82). The LP problem above (4.2) can be written in shorter form

$$\min_{\mu\in\mathcal{L}_\mathcal{G}} \langle\theta, \mu\rangle, \tag{4.3}$$

with $\langle ., . \rangle$ being the Euclidean product and μ the vector of relaxed (possibly non-integral) assignments, $\mu \in [0, 1]^{|\mathcal{V}|+|\mathcal{E}|}$.

When one minimizes the relaxed LP (4.3), model parameters θ have to be defined appropriately, depending on the specific problem application. These model parameters depend (linearly or non linearly) on so called image features and involve additionally some parameters which are usually heuristically determined. A more solid approach is to determine the parameters θ by learning. This can for instance be achieved using supervised learning, when the ground truth values μ^* (in our case labellings) are given on some training samples.

Our two learning approaches learn model parameters θ that minimize the local polytope relaxation problem in (4.3). Even though we learn approximate parameters by solving the ***relaxed*** inference problem this will not affect our prediction in a negative way as long as we use the same approximative method of inference for novel data. In fact, it was theoretically proven that this can even be a benefit, as the error in the learned approximate parameters can partly compensate for the error in the approximate inference method [Wai06]. A study on the same topic proves that learning with solving a relaxed problem (superset of the feasible set) leads to better results than learning with solving the exact problem on a subset of the feasible set [FJ08].

In general, inverse linear programming computes minimal perturbation of a given cost vector so that given the constraints, the minimizing perturbed cost vector results in a predetermined (ground truth) optimal solution. Similarly, our learning approach based on inverse linear programming, which we abbreviate from now on as **invLPA** computes perturbations θ^k for training images $k \in [N]$, of a given initial model parameter $\hat{\theta}$, obtained with any other learning method, so that the relaxed problem (4.3) attains a global minimum for each ground truth labeling μ^*, obtained via the optimization problem

$$\mu^* \in \arg\min_{\mu\in\mathcal{L}_\mathcal{G}} \langle \underbrace{\hat{\theta} + \theta^k}_{=:\tilde{\theta}^k}, \mu\rangle, \quad k \in [N] \tag{4.4}$$

for the adjusted energy $E_{\hat{\theta}+\theta}$ and the ***corrected*** model parameter $\hat{\theta} + \theta^k$. Note that we denote the corrected parameters with $\theta^k + \hat{\theta} = \tilde{\theta}^k$. For an illustrative example we refer to Fig. 4.1. We will see empirically how these corrected and approximate (due to the relaxation) parameters will result in an improvement of the final approximate prediction.

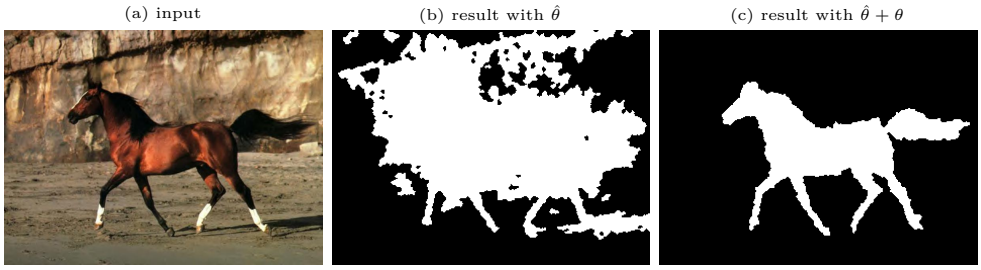


Figure 4.1. - Illustrative example of the invLPA where a perturbation to the initial model parameters (which alone results in the segmented image (b)) is estimated so that the corrected model parameter correspond (result) to the obtained (equal to ground truth) segmentation (c)

Our second approach, an instance of the first one finds linearly parametrized potentials $\theta(w)$ in terms of a parameter vector $w \in \mathbb{R}^d$, so that labellings x^k obtained by minimizing (4.3) fit ground truth labels x^{*k}. An intuitive way to model this is to learn the vector w by solving the non-convex bi-level optimization problem

$$\min_{w \in \mathbb{R}^d} \sum_{k \in [N]} \|x^k(w) - x^{*k}\|_1 \quad \text{s.t.} \quad x^k(w) \in \arg\min E_{\theta^k(w)}(x). \tag{4.5}$$

This objective expresses that we want to find parametrized potentials $\theta(w)$ so that the optimal solution would be as close as possible to the ground truth segmentation. We can think of this objective as minimizing a so called loss function which measures the difference of the estimated solution to the true one. We abbreviate this approach from now on as **LA** (linearized approach).

Our learning method based on inverse linear programming invLPA has the beneficial property that any predictor can be used to adapt model parameters to new data. In contrast, the second learning method LA is restricted to linear dependencies of the potential vectors. The linearity of the second approach is essential to come up with a tight convex relaxation that allow the usage of off-the-shelf inference solvers for labeling subproblems of the overall problem. With the invLPA method for every train data we find a corrected parameter θ^k, whereas with the LA method we learn one vector w on which all parametrized potentials for the train data depend. In particular in view of (4.5) we find one w vector on which all $x^k(w)$ depend ($\theta^k(w)$).

In this work we confine ourselves to the binary case for which the local polytope relaxation is tight if the objective function is submodular.

This chapter is organized as follows: the two new approaches for learning are presented in Sect. 4.2 and Sect. 4.3, while in Sect. 4.4 we comment on the differences of the two approaches. In Sect. 4.5 we first demonstrate emprirically the ability of invLPA to learn corrected potentials followed by empirical evaluation of both invLPA and LA on some academic examples as well as on the Weizmann horse dataset, [BU08]. In particular we demonstrate that learning benefits from inverse linear programming. In Sect. 4.6 we use our LA approach for online learning motion segmentation in video sequences, on the DAVIS video segmentation dataset [PPTM+16].

4.2. invLPA: Inverse Linear Programming Approach

Based on inverse linear programming, described in Sect. 2.6 we develop our novel approach for learning parameters in graphical models.

Our learning approach we propose in this section consists of two independent phases: (i) model parameter perturbation where we propose the novel inverse linear programming approach for learning model parameters, and (ii) model parameter prediction based on the results from (i), where we can use any model parameter prediction method.

4.2.1. Model Parameter Perturbation

In this section we apply the inverse linear programming approach of [ZL96, AO01] to the problem of learning model parameters of graphical models. For the required background on minimal representation and the exponential family model we refer to Sect. 2.3.

We consider the local polytope relaxation to the MAP inference problem, as given by

$$\min_{\mu \in \mathcal{L}_{\mathcal{G}}^{M}} \langle \hat{\theta}, \mu \rangle, \tag{4.6}$$

where the local polytope $\mathcal{L}_{\mathcal{G}}^{M}$ is defined by the so called minimal representation,

$$\mathcal{L}_{\mathcal{G}}^{M} := \begin{cases} \mu \geq 0 \\ -\mu_{ij} + \mu_i \geq 0, i \in \mathcal{V}, ij \in \mathcal{E} \\ -\mu_{ij} + \mu_j \geq 0, i \in \mathcal{V}, ij \in \mathcal{E} \\ -\mu_i - \mu_j + \mu_{ij} \geq 1, i \in \mathcal{V}, ij \in \mathcal{E}. \end{cases} \tag{4.7}$$

The second and third constraint come from the probabilistic interpretation for the binary case of two labels 0 and 1 with variables x_i, that is, $\mu_i = \mathbb{P}(x_i = 1)$, $\mu_{ij} = \mathbb{P}(x_i = 1 \wedge x_j = 1)$. In this section we will assume the minimal representation when referring to the local polytope representation. Furthermore, for compactness we will write the constraints (4.7) as a linear inequality system, i.e. $A\mu \geq b$.

In the following we will follow the inverse linear programming approach from [ZL96, AO01]. We want a ground truth segmentation, μ^* to be in the optimum of a labeling problem by perturbing an initial vector $\hat{\theta}$:

$$\mu^* \in \arg \min_{\mu \in \mathcal{L}_{\mathcal{G}}^{M}} \langle \hat{\theta} + \theta, \mu \rangle. \tag{4.8}$$

Concerning the initial model parameter vectors $\hat{\theta}$, they can be obtained with any learning method.

Our original problem (4.6) is an LP and using the results from Sect. 2.6 we can formulate the inverse LP by first constructing the dual and deriving the complementary slackness conditions. This will allow us to derive the optimal perturbation θ. In the following we denote $n := |\mathcal{V}|$ and $m := |\mathcal{E}|$.

Let us define the set of all θ parameter vectors that correspond to the ground truth segmentation μ^*

$$\Theta(\mu^*) := \{\tilde{\theta} \in \mathbb{R}^{m+n} | \min_{\mu \in \mathcal{L}_{\mathcal{G}}^M} \langle \tilde{\theta}, \mu \rangle = \langle \tilde{\theta}, \mu^* \rangle\}. \tag{4.9}$$

We want to perturb the given initial vector $\hat{\theta} \notin \Theta(\mu^*)$, in the sense of the ℓ_1 distance, i.e.

$$\theta \in \min\{||\tilde{\theta} - \hat{\theta}||_1 \, | \tilde{\theta} \in \Theta(\mu^*)\}. \tag{4.10}$$

Next we summarize the primal-dual formulation of (4.6) and use the inequality form $A\mu \geq b$ to represent the local polytope:

Primal:	**Dual:**	
$\min_{\mu} \langle \theta, \mu \rangle,$	$\max_{\nu} \langle b, \nu \rangle$	(4.11a)
s. t. $\quad A\mu \geq b, \ \mu \geq 0$	s. t. $\quad A^\top \nu \leq \theta, \ \nu \geq 0.$	(4.11b)

Let us denote with μ^* the optimal solution to the primal problem. Furthermore, let ν and ν_μ be the feasible dual variables to the optimal μ^*, corresponding to the primal constraints $A\mu \geq b$ and $\mu \geq 0$, respectively. Since we are considering a convex problem, the necessary and sufficient optimality conditions are given by the Karush-Kuhn-Tucker conditions [BV04]:

stationarity	$A^\top \nu + \nu_\mu = \theta$	(4.12a)
primal feasibility	$A\mu^* \geq b, \ \mu^* \geq 0$	(4.12b)
dual feasibility	$\nu \geq 0, \ \nu_\mu \geq 0$	(4.12c)
complementary slackness	$\langle \nu, A\mu^* - b \rangle = 0, \ \langle \nu_\mu, \mu^* \rangle = 0.$	(4.12d)

Let us define the following sets

$$\mathcal{I} := \{i \in [\dim(b)] \colon (A\mu^* - b)_i > 0\} \text{ and} \tag{4.13a}$$
$$\mathcal{J} := \{j \in [n+m] \colon \mu_j^* > 0\}. \tag{4.13b}$$

The complementary slackness conditions imply that for those indices $i \in \mathcal{I}$ we have $\nu_i = 0$ and likewise we have $(\nu_\mu)_j = 0$ for $j \in \mathcal{J}$.

Based on the primal problem in (4.11) and using the concept of inverse linear programming, see Sect. 2.6, we derive the main result of this section.

Proposition 4.2.1. *Let $n = |\mathcal{V}|, m = |\mathcal{E}|$ and $\hat{\theta} \in \mathbb{R}^{n+m}$ be a given model parameter. Suppose the local polytope based on the minimal problem representation is given by*

$$\mathcal{L}_{\mathcal{G}}^M = \{\mu \in \mathbb{R}_+^{n+m} \colon A\mu \geq b\}. \tag{4.14}$$

Let $\mu^* \in \{0,1\}^{n+m}$ be a given binary ground truth labeling. Then the minimal

ℓ_1-norm perturbation $\theta \in \mathbb{R}^{n+m}$ of $\hat{\theta}$ is such that $\mu^* \in \arg\min\{\langle \hat{\theta} + \theta, \mu \rangle \colon \mu \in \mathcal{L}_{\mathcal{G}}^M\}$ is a solution to the linear program

$$\min_{\theta, \nu_\mu, \nu} \|\theta\|_1 \quad s.t. \quad A^\top \nu + \nu_\mu = \hat{\theta} + \theta \tag{4.15a}$$

$$\theta \in \mathbb{R}^{n+m},\ \nu_\mu \in \mathbb{R}_+^{n+m},\ \nu \in \mathbb{R}_+^{\dim(b)},\ \nu_{\mathcal{I}} = 0,\ (\nu_\mu)_{\mathcal{J}} = 0, \tag{4.15b}$$

$$\mathcal{I} := \{i \in [\dim(b)] \colon (A\mu^* - b)_i > 0\},\ \mathcal{J} := \{j \in [n+m] \colon \mu_j^* > 0\}. \tag{4.15c}$$

Proof. Having initial feasible $\hat{\theta}$ we want to adjust it (correct it) so that the final $\hat{\theta} + \theta$ corresponds to the optimal solution μ^* to the primal problem in (4.11). From the theory of inverse linear programming [ZL96, AO01] we know that this amounts to solving an LP (4.10). On the other hand the necessary KKT optimality conditions are sufficient, too, since we have a linear (convex) function. From this follows that the minimal norm perturbation θ can be found by replacing θ by $\hat{\theta} + \theta$ in (4.12), which then leads to the constraints in (4.15). □

The problem in (4.15) is not a linear program in the above written form but can be easily converted into one:

$$\min_{\nu, \nu_\mu \geq 0, \theta} ||\theta||_1 \quad \text{s.t.} \tag{4.16a}$$

$$A^\top \nu + \nu_\mu - \theta = \hat{\theta} \tag{4.16b}$$

$$D_1(\mu^*)\nu = 0 \tag{4.16c}$$

$$D_2(\mu^*)\nu_\mu = 0 \tag{4.16d}$$

where D_1 and D_2 are the diagonal matrices that correspond to the constraints from (4.15), i.e.

$$D_1(\mu^*)_{ii} := \begin{cases} 1, & \text{if } i \in \mathcal{I} \\ 0, & \text{otherwise} \end{cases} \qquad i \in [\dim(b)] \tag{4.17a}$$

$$D_2(\mu^*)_{jj} := \begin{cases} 1, & \text{if } j \in \mathcal{J} \\ 0, & \text{otherwise} \end{cases} \qquad j \in [n+m]. \tag{4.17b}$$

The equation (4.16) above is equivalent to the linear program

$$\min_{\nu, \nu_\mu, \theta^+, \theta^- \geq 0} \langle \mathbb{1}, \theta^+ + \theta^- \rangle \quad \text{subject to} \tag{4.18a}$$

$$\begin{pmatrix} A^\top & I_{n+m} & -I_{n+m} & I_{n+m} \\ D_1(\mu^*) & 0 & 0 & 0 \\ 0 & D_2(\mu^*) & 0 & 0 \end{pmatrix} \begin{pmatrix} \nu \\ \nu_\mu \\ \theta^+ \\ \theta^- \end{pmatrix} = \begin{pmatrix} \hat{\theta} \\ 0 \\ 0 \end{pmatrix} \tag{4.18b}$$

where $\theta^+ = \max\{\theta, 0\}$ and $\theta^- = -\min\{\theta, 0\}$. The linear program above can be solved with a linear programming solver, for example MOSEK, [ApS15].

4.2.2. Model Parameter Prediction

In the previous section we saw how to compute the perturbation potentials θ, given the ground truth labelings, μ^*. In this section we describe the second phase of invLPA, which is completely independent of the first one. Here we use the perturbation potentials computed in the first phase to predict new potentials based on novel data features. Our training data comprises the learned perturbed potentials $\tilde{\theta}^k = \hat{\theta} + \theta^k$, and corresponding features f^k (unary or pairwise), $k \in [N]$. Note that the outcome from the invLPA (4.18) is a vector $\theta = (..., \theta^k, ...)$ where θ^k is a scalar value which is the perturbation value for the corresponding node k (for the case for learning unary potentials).

We are free to choose any model prediction method that returns model parameter vector θ based on observed novel features. However, in this work we limit ourselves to simple linear prediction methods and a nonlinear (NL) Gaussian regression method able to capture a richer model structure.

Linear Prediction

For linear prediction we consider two common methods, linear least-squares (LS), and a sparse ℓ_1-norm approach. Furthermore, we assume a linear dependency of our potentials on a vector w, i.e.

$$\theta^k = \langle f^k, w \rangle, \quad k \in [N] \tag{4.19}$$

where $f^k, w \in \mathbb{R}^{\dim f^k}$. Here θ^k are the potentials of a discrete graphical model, see (4.1), and can be either unary or pairwise ones. Furthermore, f^k represent the corresponding features. In the next step, the perturbed model parameters $\tilde{\theta}^k = \hat{\theta} + \theta^k$, $k \in [N]$, obtained by solving (4.15) are fit to the observed features f^k. To this end we collect all our corrected model parameters learned in the first phase and fit linearly parametrized model parameters to them. We do this with two linear fitting methods as described next.

Least-squares fitting: We set up an overconstrained system and solve

$$\min_w \sum_{k \in [N]} |\langle f^k, w \rangle - \tilde{\theta}^k|^2. \tag{4.20}$$

Please keep in mind that in the equation above the features f^k are the corresponding ones to the perturbed potentials $\tilde{\theta}^k$.

ℓ_1-norm fitting: In addition to the smooth least-squares approach, we also apply the sparse regularization approach

$$\min_{w, s_k} \|w\|_1 + \lambda \sum_{k \in [N]} |s_k|, \qquad \text{s.t. } \langle f^k, w \rangle - s_k = \tilde{\theta}^k, k \in [N]. \tag{4.21}$$

where $\lambda > 0$ is some parameter. Here again f^k are the features corresponding to $\tilde{\theta}^k$ in the training data.

After having found one fitting vector w for all train data we can use it, together with the novel features on the test data, to construct new predicted linearized model parameter vectors as in (4.19).

Nonlinear Prediction

To demonstrate the flexibility of our method we apply different model prediction methods. For this reason we use a nonlinear Gaussian regression, [RW06], which can capture more from the model structure as opposed to simple linear methods. The Gaussian prediction model for obtaining prediction potentials $\overline{\theta}(f)$ is given in the form

$$\overline{\theta}(f) := k_N(f)^\top \big(K(F) + \sigma_n^2 I\big)^{-1}\tilde{\theta} \qquad (\sigma_n^2 \text{ is a parameter}) \tag{4.22a}$$

$$= \sum_{k\in[N]} w_k(F,\tilde{\theta})k(f^k,f), \qquad w(F,\tilde{\theta}) := \big(K(F) + \sigma_n^2 I\big)^{-1}\tilde{\theta}, \tag{4.22b}$$

where $K(F)$ is the covariance matrix induced by the training data, that is

$$K(F) = \{k(f^k,f^l)\}_{k,l\in[N]} \tag{4.23}$$

where

$$k(f^k,f^l) := \sigma_m^2 \exp\Big(-\frac{1}{2\sigma_f^2}\|f^k - f^l\|^2\Big), \qquad (\sigma_f^2,\sigma_m^2 \text{ are parameters}), \tag{4.24}$$

and

$$k_N(f) := \big(k(f^1,f),\ldots,k(f^N,f)\big)^\top \tag{4.25}$$

evaluates for any novel feature vector f the kernel function using all given feature vectors f^k, $k \in [N]$, for the training data. Thus, given a novel image with feature vector f, the corresponding model parameter is $\overline{\theta} = \overline{\theta}(f)$.

One drawback of a Gaussian regression is its cubic complexity. However, in the literature there are many solutions suggested to this problem, all finding an approximation to the whole model, e.g. by using a sparse Gaussian regression and selecting a subset of the training data by random. Another option is approximation of the matrix $K(F)+\sigma_n^2 I$ with a low rank matrix. In contrast, the Bayesian Committee machine[Tre00, ST02] splits the whole data into random subsets (clusters) and predicts subset-wise, while considering the testing data when making a prediction.

An approach similar to the Bayesian Committee machine was proposed in [ND14], which considers the inductive property instead of the transductive one. The training data is split by random into smaller subsets and exact inference is performed on every of these subsets. In this way the computation time can be reduced if we can parallelize the inference. In addition, the complexity can be significantly reduced. In fact the complexity is now linear in the number of data N. If we consider splitting into M subsets, then we have to invert an N/M matrix and we can write $M = N/\alpha$, for some scalar α.

4.3. LA: Linearly Parametrized Joint Learning Approach

In this section we develop our second method, the linearly parametrized approach (LA), that jointly determines model parameter perturbations and predictions based on linearly parametrized potential functions. The motivation for this alternative is that when using linearly parametrized potentials we can derive a tight convex relaxation and are able to use off-the-shelf inference implementations for solving the labeling subproblems of the overall learning model.

We first work out a relaxation of the original learning problem (4.5) and complement it by an appropriate optimization method. For this approach we use the usual non-minimal or overcomplete representation which allows us to resort to established solvers for labeling problems in subroutines. More precisely, we index θ and μ as $\theta^i(x_i)$, $\theta^{ij}(x_i, x_j)$, $\mu_i(x_i)$, $\mu_{ij}(x_i, x_j)$, $i \in \mathcal{V}$, $ij \in \mathcal{E}$, by binary labellings $x_i \in \{0, 1\}$, $i \in \mathcal{V}$ and the usual local polytope constraints from (2.82).

This approach jointly finds the perturbed approximate model parameters (as we consider a relaxed problem) and fits them to newly observed data, leading to a set of predicted (updated) model parameters. For clarity, we split the two phases when deriving the method.

4.3.1. Model Parameter Perturbation

For deriving the final relaxed problem we will explore the dual of the local polytope relaxation of the labeling problem (4.3). We start by formulating the primal-dual pair of LPs:

$$\begin{aligned}
&\textbf{Primal:} && \textbf{Dual:} && (4.26a)\\
&\min_{\mu} \langle \theta, \mu \rangle \ \text{ s.t.} && \max_{\psi,\phi} \psi \ \text{ s.t} && (4.26b)\\
&\sum_{x_j} \mu_{ij}(x_{ij}) - \mu_i(x_i) = 0 && \psi_{ij}(x_i) \in \mathbb{R} && (4.26c)\\
&\sum_{x_i} \mu_{ij}(x_{ij}) - \mu_j(x_j) = 0 && \psi_{ji}(x_j) \in \mathbb{R} && (4.26d)\\
&\sum_{x_i} \mu_i(x_i) - \mu_0 = 0 && \phi \in \mathbb{R}^{\dim \mu} && (4.26e)\\
&\mu_0 = 1 && \psi \in \mathbb{R} && (4.26f)\\
&\mu_i(x_i) \geq 0 && \theta^i(x_i) + \sum_{j \in \mathcal{N}(i)} \psi_{ij}(x_i) - \phi_i \geq 0 && (4.26g)\\
&\mu_{ij}(x_{ij}) \geq 0 && \theta^{ij}(x_{ij}) + \phi_{ij}(x_i) + \phi_{ji}(x_j) \geq 0 && (4.26h)\\
&\mu_0 \geq 0 && \theta^0 + \sum_{i \in \mathcal{V}} \phi_i - \psi \geq 0 && (4.26i)
\end{aligned}$$

or more compactly

$$\begin{aligned} &\textbf{Primal:} && \textbf{Dual:} && (4.27a)\\ &\min_{\mu\geq 0}\langle\theta,\mu\rangle, && \max_{\psi,\phi}\psi && (4.27b)\\ &\text{s.t}\quad A\mu=0,\ \mu_0=1 && \text{s.t}\quad \theta - A^\top\phi - e_0\psi \geq 0. && (4.27c)\end{aligned}$$

Note that the primal vector μ is augmented by the scalar variable μ_0 as first component which enforces the mass constraints $\sum_{x_i\in\{0,1\}}\mu_i(x_i)-\mu_0=0, i\in\mathcal{V}$ as part of the system $A\mu=0$ which enforces the local polytope constraint (2.82). The vector ϕ together with ψ form the dual variables [1]. Furthermore, $e_0=(1,0,...,0)^\top$ is the unit basis vector. It becomes clear that the primal formulation is equivalent to (4.3) with the local polytope constraints from (2.82) in a slightly different form. The dual formulation is just the reparametrized formulation of the relaxed labeling problem over the local polytope, [Wer07]. Accordingly we define the reparametrized potentials as

$$\theta_\phi := \theta - A^\top\phi. \tag{4.28}$$

Let μ^* be a solution of the primal problem. The complementary slackness conditions state that there exists a pair (ψ,ϕ) of dual variables such that

$$\begin{aligned} \mu_i^*(x_i)>0 \quad &\Longrightarrow \quad \theta_\phi^i(x_i)=0, && (4.29a)\\ \mu_{ij}^*(x_i,x_j)>0 \quad &\Longrightarrow \quad \theta_\phi^{ij}(x_i,x_j)=0, && (4.29b)\\ \mu_0^*>0 \quad &\Longrightarrow \quad \psi=\theta^0+\sum_{i\in\mathcal{V}}\phi_i. && (4.29c)\end{aligned}$$

The third implication can be always satisfied if ψ is set accordingly. We want to have a ***unique*** optimal pair μ^* and x^* corresponding to the ground truth labeling. For this reason and to meet the implications (4.29a) and (4.29b), we choose

$$\begin{aligned} \theta_\phi^i(x_i^*)&=0,\\ \theta_\phi^i(1-x_i^*)&\geq\varepsilon, \end{aligned} \quad \text{and} \quad \begin{aligned} \theta_\phi^{ij}(x_i^*,x_j^*)&=0,\\ \theta_\phi^{ij}(x_i,x_j)&\geq\varepsilon, \quad \forall(x_i,x_j)\neq(x_i^*,x_j^*), \end{aligned} \tag{4.30}$$

for some $\varepsilon>0$ and all $i\in\mathcal{V}$, $ij\in\mathcal{E}$. Introducing the ε term is crucial, since this choice guarantees a unique optimal θ_ϕ.

For this approach we restrict our model parameters to depend linearly on the feature vector f and a vector $w\in\mathbb{R}^{\dim f}$, i.e. $\theta=(\theta^i,\theta^{ij})^\top$ linearly depends on $w=(w_u,w_p)^\top$ and we can rewrite the model parameters as,

$$\theta_{w,\phi} := \theta(w) - A^\top\phi. \tag{4.31}$$

Using an additional vector $s\geq 0$ of slackness variables in order to convert the inequalities in (4.30) into equalities, we obtain the relaxed formulation in the form

[1] We do not denote the dual variables by ν, as in the preceding section, due to the slightly different LP formulation (4.27)

of a linear program

$$\min_{w,\phi,s\geq 0} \langle \mu^*, s\rangle, \qquad \langle \mu^*, s\rangle = \sum_{i\in\mathcal{V}} s_i(x_i^*) + \sum_{ij\in\mathcal{E}} s_{ij}(x_i^*, x_j^*) \tag{4.32a}$$

$$\text{s.t.} \quad \begin{cases} \theta^i_{w,\phi}(x_i^*) - s_i(x_i^*) = 0, \\ \theta^i_{w,\phi}(1 - x_i^*) - s_i(1 - x_i^*) = \varepsilon, \\ \theta^{ij}_{w,\phi}(x_i^*, x_j^*) - s_{ij}(x_i^*, x_j^*) = 0, \\ \theta^{ij}_{w,\phi}(x_i, x_j) - s_{ij}(x_i, x_j) = \varepsilon, \quad \forall (x_i, x_j) \neq (x_i^*, x_j^*), \end{cases} \tag{4.32b}$$

for $\varepsilon > 0$ and all $i \in \mathcal{V}$, $ij \in \mathcal{E}$. The choice of ε will be discussed in detail after deriving the final relaxed problem. Optionally, we can add an additional convex constraint in (4.32), $\langle w_p, f^{ij}\rangle \geq 0$, where f^{ij} are the predefined pairwise data features. This constraint can be added for enforcing submodularity, see Definition 2.4.2, which allows solving in polynomial time. Alternatively, quadratic pseudo-boolean optimization (QPBO) solvers, see [KR07] and Sect. 2.4.4, can be used without the additional submodularity constraint but without a guarantee for a complete labeling. However, unassigned nodes can be assigned any label without having a negative aspect on the energy due to the persistency property of the QPBO, see Sect. 2.4.4.

Regarding the formulation (4.32) above we aim at penalizing the slackness variables s. That is the smaller the objective is, the closer we are to the unique optimal solution, the ground truth segmentation. When $s = 0$ this implies that the objective is zero, which means we have obtained the ground truth segmentation.

4.3.2. Model Parameter Prediction

The large scale LP in (4.32) can not be solved with standard LP solvers due to the large problem sizes typical for Markov random fields (MRF) occurring in computer vision applications. Therefore, in this section we propose splitting the task into a labeling and a parameter estimation subproblem. The labeling subproblems can then be solved with available max-flow solvers if the submodularity constraint, $\langle w_p, f^{ij}\rangle \geq 0$ is included. The general case can be handled by a QPBO solver [KR07]. The subproblem solutions are fused using Lagrange multipliers. In order to derive the formulation of the decomposed problem, we consider linearly parametrized local potentials of the form

$$\theta^i(w; x_i) = \begin{pmatrix} \theta^i(w;0) \\ \theta^i(w;1) \end{pmatrix} = \begin{pmatrix} \langle w_u^0, f^i\rangle \\ \langle w_u^1, f^i\rangle \end{pmatrix}, \tag{4.33a}$$

$$\theta^{ij}(w; x_i, x_j) = \begin{pmatrix} \theta^{ij}(w;0,0) & \theta^{ij}(w;0,1) \\ \theta^{ij}(w;1,0) & \theta^{ij}(w;1,1) \end{pmatrix} = \begin{pmatrix} \langle w_p^{00}, f^{ij}\rangle & \langle w_p^{01}, f^{ij}\rangle \\ \langle w_p^{10}, f^{ij}\rangle & \langle w_p^{11}, f^{ij}\rangle \end{pmatrix}, \tag{4.33b}$$

where f^i, f^{ij} denote arbitrary unary and pairwise feature vectors extracted at pixel $i \in \mathcal{V}$ and edge $ij \in \mathcal{E}$. In the formulation above we consider the vector $w = (w_u, w_p)^\top$ as

$$w = \begin{pmatrix} w_u^0 & w_u^1 & w_p^{00} & w_p^{01} & w_p^{10} & w_p^{11} \end{pmatrix}^\top. \tag{4.34}$$

For simpler reformulation of (4.32) we define modified cost potentials

$$\tilde{\theta}^{i}(w;x_i) := \begin{pmatrix} \langle w_u^0, f^i\rangle - \varepsilon \mathbb{1}_{\{x_i^*=1\}} \\ \langle w_u^1, f^i\rangle - \varepsilon \mathbb{1}_{\{x_i^*=0\}} \end{pmatrix}, \tag{4.35a}$$

$$\tilde{\theta}^{ij}(w;x_i,x_j) := \begin{pmatrix} \langle w_p^{00}, f^{ij}\rangle - \varepsilon \mathbb{1}_{\{(x_{ij}^*)\neq(0,0)\}} & \langle w_p^{01}, f^{ij}\rangle - \varepsilon \mathbb{1}_{\{(x_{ij}^*)\neq(0,1)\}} \\ \langle w_p^{10}, f^{ij}\rangle - \varepsilon \mathbb{1}_{\{(x_{ij}^*)\neq(1,0)\}} & \langle w_p^{11}, f^{ij}\rangle - \varepsilon \mathbb{1}_{\{(x_{ij}^*)\neq(1,1)\}} \end{pmatrix}, \tag{4.35b}$$

with $\mathbb{1}_{\text{predicate}} = 1$ if predicate is true and $\mathbb{1}_{\text{predicate}} = 0$ otherwise. Based on the new redefined reparametrized potentials in (4.35), and in view of (4.31) we can rewrite (4.32) as

$$\min_{w,\phi,s} \langle \mu^*, s\rangle \quad \text{subject to} \quad \tilde{\theta}_{w,\phi} - s = 0\,. \tag{4.36}$$

Now in order to solve this problem we fix w and thus the reparametrized potentials $\tilde{\theta}_w$ and solve the labeling problem in s and ϕ. Due to the large scale character of this problem, we derive its dual formulation which is more efficient to solve.

Primal:	**Dual:**	
$\min_{\phi,s} \langle \mu^*, s\rangle$ s.t.	$\max_{\mu} \langle -\tilde{\theta}_w, \mu\rangle$ s.t.	(4.37b)
$\tilde{\theta}_{w,\phi}(x_i) - s(x_i) = 0$	$\mu_i(x_i) \in \mathbb{R}$	(4.37c)
$\tilde{\theta}_{w,\phi}(x_i,x_j) - s(x_i,x_j) = 0$	$\mu_{ij}(x_i,x_j) \in \mathbb{R}$	(4.37d)
$\phi_{ij}(x_i) \in \mathbb{R}$	$\sum_{x_j\in\{0,1\}} \mu_{ij}(x_i,x_j) - \mu_i(x_i) = 0$	(4.37e)
$\phi_{ji}(x_j) \in \mathbb{R}$	$\sum_{x_i\in\{0,1\}} \mu_{ij}(x_i,x_j) - \mu_j(x_j) = 0$	(4.37f)
$s_i(x_i) \geq 0$	$\mu_i(x_i) - \mu_i^*(x_i) \geq 0$	(4.37g)
$s_{ij}(x_i,x_j) \geq 0$	$\mu_{ij}(x_i,x_j) - \mu_{ij}^*(x_i,x_j) \geq 0.$	(4.37h)

The constraints are initiated for each node $i \in \mathcal{V}$ and edge $ij \in \mathcal{E}$. Note that in the dual we have $\tilde{\theta}_w$ without the ϕ part as in (4.31). Also the constraints for the dual do not correspond to the usual form of local polytope constraints (2.82). However, we can substitute μ by $\mu - \mu^*$ yielding $\mu_i(x_i) \geq 2\mu_i^*(x_i) \geq 0$ for (4.37g) and similar to (4.37h), which now fit the local polytope form. Furthermore, we have to add $\langle \tilde{\theta}_w, \mu^*\rangle$ to the dual objective, which for fixed w is indeed a constant value.

We are now ready to formulate the final saddle point problem, reformulation of the dual from (4.37) taking into account a collection of N training samples

$$\min_w L(w), \qquad L(w) = \sum_{k\in[N]} \max_{\mu\in\mathcal{L}_{\mathcal{G}}} \{\langle -\tilde{\theta}_w, \mu\rangle + \langle \tilde{\theta}_w, \mu_k^*\rangle\}. \tag{4.38}$$

Remark 4.3.1. Adding a convex regularization term with respect to w, like $\frac{1}{2}||w||^2$ (or only $\frac{1}{2}||w_u||^2$ or $\frac{1}{2}||w_p||^2$) to the objective function (4.38) may be beneficial to avoid overfitting in a high-dimension parametrization.

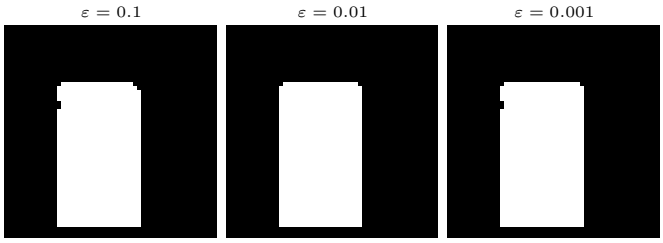


Figure 4.2. **-** Resulting segmentation after learning with different choices of ε. We can see that $\varepsilon = 0.01$ produces the best segmentation or smallest error in the sense of percentage of mislabeled pixels, 0.06%, whereas the error with $\varepsilon = 0.1$ and $\varepsilon = 0.001$ is 0.19% and 0.13% respectively.

Setting the parameter ε

The parameter ε was introduced in (4.30) in order to ensure uniqueness of the optimal pair μ^* and x^* or a unique solution to the primal problem in (4.27). In our learning ε is a parameter to be set up by hand carefully, taking into account the range of the unary and pairwise features, f^i and f^{ij} in (4.35). In the general case ε should be chosen such that $0 < \varepsilon \ll f^i$ and $0 < \varepsilon \ll f^{ij}$. In our experiments we usually normalize the features to be in the range $[0, 1]$, so we need to set ε so that $0 < \varepsilon \ll 1$.

We examine empirically how several choices of ε affect the learning. We take a simple example for learning unary terms while fixing pairwise terms to be Ising prior with value 1. Details on the experimental set up will be discussed in Sect. 4.5.2. We use the same image for training and testing and report here the results in terms of ε in Fig. 4.2.

From the experiment above we can see that as long as $0 < \epsilon \ll 1$ the segmentation results do not change significantly. As long as ε is chosen to be smaller than the feature vectors it does not influence much on the final segmentation.

4.3.3. Optimization

Minimizers of the labeling problems form subgradients of the parameter learning problem, see Sect. 2.2.4 for a brief overview on subgradient methods. Let μ^{w_0} be a solution to a labeling subproblem of (4.38) for a fixed w_0, which can be obtained with an inference solver, e.g. QPBO, [KR07], see Sect. 2.4.4. The subgradients with

respect to $w = (w_u, w_p)$ in the form (4.34) are given by

$$\partial_{w_u} L(w) = \sum_{k\in[N]} \sum_{i\in\mathcal{V}^k} \Big(f^{k,i} \cdot (-\mu_{k,i}^{w_0}(0) + \mu_{k,i}^*(0)),\ f^{k,i} \cdot (-\mu_{k,i}^{w_0}(1) + \mu_{k,i}^*(1)) \Big)^\top \tag{4.39a}$$

$$\partial_{w_p} L(w) = \sum_{k\in[N]} \sum_{ij\in\mathcal{E}^k} \Big(f^{k,ij} \cdot (-\mu_{k,ij}^{w_0}(0,0) + \mu_{k,ij}^*(0,0)), \tag{4.39b}$$

$$f^{k,ij} \cdot (-\mu_{k,ij}^{w_0}(0,1) + \mu_{k,ij}^*(0,1)), \tag{4.39c}$$

$$f^{k,ij} \cdot (-\mu_{k,ij}^{w_0}(1,0) + \mu_{k,ij}^*(1,0)),\ f^{k,ij} \cdot (-\mu_{k,ij}^{w_0}(1,1) + \mu_{k,ij}^*(1,1)) \Big)^\top. \tag{4.39d}$$

The index k denotes the training image number, $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$ the corresponding graph, and $f^{k,i}$ and $f^{k,ij}$ the features of the corresponding nodes and edges. Furthermore, $\mu_{k,i}^{w_0}$ and $\mu_{k,ij}^{w_0}$ are the label assigned to node i and edge ij respectively, and similarly, $\mu_{k,i}^*$ and $\mu_{k,ij}^*$ are the ground truth labels.

Note that the features are vectors and $\mu_{k,i}$, $\mu_{k,ij}$ and μ_i^*, μ_{ij}^* are scalars. The products of the feature vectors $f^{k,i}$ and $f^{k,ij}$ with $\mu_{k,i} + \mu_{k,i}^*$ and $\mu_{k,ij} + \mu_{k,ij}^*$ respectively result in a vector with the same dimension as the feature vector. Consequently, for the dimension of the subgradients are $\partial_{w_u} L(w) \in \mathbb{R}^{2\dim f^i}$ and $\partial_{w_p} L(w) \in \mathbb{R}^{4\dim f^{ij}}$. We denote that this is the general case when representing the parameters θ as in (4.33). However, for some of the experiments in Sect. 4.5 we will consider w with less parameters.

If considering additionally the submodularity constraint $\langle w_p, f^{ij} \rangle \geq 0$, we need to use a projected subgradient method. However, the projection to a set with such a high number of edges is computationally expensive. For this reason we will exclude this constraint and can resort to a subgradient method without projections.

In general, subgradient methods are extremely slow and for this reason we will use a smoothed version of the subgradient method, also known in the literature as a deflected subgradient method see e.g. [dF09, Gut03, CFM75]. It is often combined with a (modified) Polyak step size rule [Pol69], which leads to a significant speed up of the subgradient method. In Algorithm 3 we present the applied optimization algorithm in pseudo code form. Note that for clarity we consider only the case of a single training image and re-use index k as the iteration index. Furthermore, g^k and f^k denote the general and the smoothed subgradient in iteration k. $\overline{L}$ stands for the optimal (upper bound) value of the objective we do not actually know. A detailed discussion of the algorithm and its convergence follows in the next section.

The algorithm described in 3 combines two strategies for enhancing the basic subgradient method:

1. We refrain from using the common 'divergent series' approach with a diminishing step size α_k satisfying $\sum_{k\geq 0} \alpha_k = \infty$, that may converge very slowly. Instead we apply the adaptive step size $\frac{1}{\|g^k\|^2}(L(w^k) - \overline{L})$ due to [Pol69], where $\overline{L}$ is adjusted in line (4-5) and (15-16) in order to compensate for not knowing the optimal value L^*.

Algorithm 3: Deflected Subgradient Algorithm with a Modified Polyak Step Size

input : $k = 1$, $w^1 = 1$, $f^0 = 0$, $\overline{L} = 0.1$, $L(w^1) = \infty$, $\epsilon_1 = 10^{-5}$, $\epsilon_2 = 10^{-6}$,$n = 40$

while $L(w^k) - \overline{L} > \epsilon_2$ **do**
- Obtain optimal labeling $\mu(w^k)$ with max-flow solver (for instance QPBO solver [KR07] in our case) and update the objective $L(w^k)$;
- $L_{min} = \min_{i=1,\ldots,k}\{L(w^i)\}$;
- **if** $L(w^k) < \overline{L}$ **then**
  - $\overline{L} = L(w^k) - \epsilon_1$;
- $g^k \in \partial_w L(w^k)$;
- $\beta^k = \max\{0, -1.5\frac{g^k(f^{k-1})^\top}{||f^{k-1}||^2}\}$;
- $f^k = g^k + \beta^k f^{k-1}$;
- **if** $||f^k|| < \epsilon_2$ **then**
  - break;
- $\alpha_k = \frac{L(w^k)-\overline{L}}{||f^k||^2}$;
- $\hat{w}^{k+1} = w^k - \alpha_k f^k$;
- $w^{k+1} = P_S(\hat{w}^{k+1})$;
- $k = k + 1$;
- **if** $\mod(k, n) = 0 \wedge L_{min} \geq \overline{L} + \epsilon_1$ **then**
  - $\overline{L} = \frac{\overline{L}+L_{min}}{2}$;

return w^k

2. The sequence of subgradients $(g^k)_{k\geq 0}$ is replaced by $(f^k)_{k\geq 0}$, see line (8) as suggested by [CFM75], to obtain a smoother sequence of updates $(w^k)_{k\geq 0}$ in line (12) and faster convergence.

Remark 4.3.2. We include the projection step in line (13) of Algorithm 3 for generality. In particular it allows to use max-flow solvers other than QPBO, which require submodular energy, 2.4.2. However in our implementation we use QPBO and omit the projection step.

4.3.4. Convergence Analysis of the Deflected Subgradient Method with a Modified Polayk Step Size

The algorithm presented in the previous section will be used for optimizing (4.38) and thus it is of great interest to analyse its convergence properties.

Ordinary subgradient method can be very slow as the step direction may not be a descent direction and the sub-optimal choice of the step size.

While for an ordinary subgradient method in every iteration any previous information is discarded and not used, deflected subgradient methods are based on combining a filtered information from the previous iterate with the present one. In this sense the so called zigzagging phenomenon of the subgradient method which can also slow down the process can be curred. However, when using the projected subgradient a different type of zigzagging might occur during iterations near the feasible set boundary. The reason for this type of zigzagging is that the feasible set is not considered when forming the direction. In order to remove both types of zigzagging two types of deflection should be combined in a proper way. This is worked out in [dF09].

The deflected subgradient in Algorithm 3 corrects only the first type of zigzagging and since we want to prove its convergence we concentrate on this type of deflection methods.

A subgradient of a convex lower semi-continuous function L, at an iterate w^k is a vector g^k satisfying

$$\langle g^k, w^k - w\rangle \geq L(w^k) - L(w) \quad \forall w \in \operatorname{dom} L. \tag{4.40}$$

The step size is of importance for convergence of the subgradient method. The usual step size rules, which guarantee convergence like the square summable and divergent sequences step sizes are very slow in practice. Another choice is the Polyak step size [Pol69] which also guarantees convergence, and on the other hand was found to be much faster in practice. It is given by

$$\alpha_k = \frac{L(w^k) - L^*}{||g^k||^2} \tag{4.41}$$

where L^* is the optimal function value and g^k the subgradient (4.40). Note that for deflected subgradient method we use f^k, see (4.44) instead of g^k. However in most of the cases the optimal value L^* is not known. Then one can use a modified Polyak

step size where L^* is replaced with an estimate $\overline{L}$ which can either be an upper or a lower bound on L^*.

Remark 4.3.3. In our Algorithm 3 line (4-5) we make sure that the value $\overline{L}$ is such that the step size α_k in (4.41) with $L^* = \overline{L}$ is always larger than zero. This is an important property of the step size for the subgradient method which should be satisfied, see Proposition 2.2.5.

We need to prove the extension of the subgradient properties to the deflected subgradient method (in combination with the modified Polyak step size) which corrects the subgradient direction and causes the iterates to move faster in the direction of the optimal value iterate. Let us consider the sequence of iterates as in the Algorithm 3,

$$\hat{w}^{k+1} = w^k - \alpha_k f^k \tag{4.42}$$

and

$$w^{k+1} = P_S(\hat{w}^{k+1}) \tag{4.43}$$

with S the set to be projected on, which is closed and convex. Our deflected subgradient is given by

$$f^k = g^k + \beta^k f^{k-1} \tag{4.44}$$

with β^k chosen as in line (7) in Algorithm 3. We will prove that the deflected subgradient direction is a subgradient direction, too, from where would follow that the properties for the subgradient direction can be extended to the deflected direction. Moreover we will additionally prove convergence for the deflected subgradient method when the step size is the modified Polyak step size.

First we prove convergence for subgradient method with modified Polyak step size when an upper bound is used instead of the optimal function value. We use the result from [Ned08, Theorem 4] and adapt it to our step size choice.

Lemma 4.3.1. ***[Convergence of subgradient method with modified Polyak step size]*** *Let the projection set S be closed and convex, and the function L be convex over $\mathbb{R}^n$ with finite optimal value L^*. Let the set S^* of optimal w^* values be nonempty. Then for the iterative sequence $\{w^k\}$ generated by the subgradient method with Polyak step size*

$$\alpha_k = \frac{L(w^k) - \overline{L}}{||g^k||^2} \tag{4.45}$$

where $\overline{L} \geq L^*$ is an upper bound of the optimal value L^* and g is the subgradient given by (4.40) we have

$$\lim_{k\to\infty} ||w^k - w^*|| = 0 \quad \textit{for some } w^* \in S^*. \tag{4.46}$$

Proof. First let $y \in S$ be arbitrary fixed and let α_k be any step size. Then for all k

$$||w^{k+1} - y||^2 \leq ||w^k - \alpha_k g^k - y||^2 \tag{4.47a}$$
$$= ||w^k - y||^2 - 2\alpha_k (g^k)^\top (w^k - y) + \alpha_k^2 ||g^k||^2 \tag{4.47b}$$
$$\leq ||w^k - y||^2 - 2\alpha_k (L(w^k) - L(y)) + \alpha_k^2 ||g^k||^2. \tag{4.47c}$$

The first inequality comes from (4.42) and the non expansive property of a projection. The last inequality follows from the subgradient property (4.40). The same inequality will hold for $y = w^* \in S$. Then plugging in the step size α_k from (4.45) in (4.47) and using $\overline{L} \geq L^*$ we have

$$||w^{k+1} - w^*||^2 \leq ||w^k - w^*||^2 - \frac{(L(w^k) - L^*)^2}{||g^k||^2}. \tag{4.48}$$

By applying the inequality (4.48) recursively, it follows that for any $w^* \in S^*$ and any k and l with $k > l$,

$$||w^{k+1} - w^*||^2 \leq ||w^l - w^*||^2 - \sum_{i=l}^{k} \frac{(L(w^i) - L^*)^2}{||g^i||^2}. \tag{4.49}$$

With $l = 0$, the iterate sequence $\{w^k\}$ is bounded and consequently has an accumulation point. From (4.49)

$$\sum_{i=0}^{\infty} \frac{(L(w^i) - L^*)^2}{||g^i||} \leq ||w^0 - w^*||^2 < \infty. \tag{4.50}$$

Let us suppose that none of the accumulation points of $\{w^k\}$ belongs to S^*. Then for a small $\epsilon > 0$

$$L(w^k) > L^* + \epsilon \quad \text{for all } k. \tag{4.51}$$

From the boundedness of $\{w^k\}$, it follows that for $s^k \in \partial L$ for all k the sequence $\{s^k\}$ is bounded too and every of its limit points is a subgradient. Then the sequence of subgradients $\{g^k\}$ is bounded too, that is there exists some $m > 0$ so that

$$||g^k|| \leq m \quad \text{for all } k. \tag{4.52}$$

Then

$$\frac{(L(w^i) - L^*)^2}{||g^i||^2} \geq \frac{\epsilon^2}{m^2}. \tag{4.53}$$

By summing this equation over i we get that

$$\sum_{i=0}^{\infty} \frac{(L(w^i) - L^*)^2}{||g^i||^2} \geq \sum_{i=0}^{\infty} \frac{\epsilon^2}{m^2} = \infty. \tag{4.54}$$

This is a contradiction to (4.50), so our assumption was wrong which implies that every accumulation point of $\{w^k\}$ must belong to the set S^*. Let $\hat{w}^*$ be an accumulation

point of the sequence $\{w^k\}$ and let $\{w^{k_j}\}$ be a subsequence of $\{w^k\}$ converging to the accumulation point $\hat{w}^*$. With setting $w^* = \hat{w}^*$ and $l = k_j$ in (4.49) we obtain for all $k > k_j$

$$||w^{k+1} - \hat{w}^*||^2 \leq ||w^{k_j} - \hat{w}^*||^2 - \sum_{i=k_j}^{k} \frac{(L(w^i) - L^*)^2}{||g^i||^2}. \tag{4.55}$$

From here it follows

$$\lim_{k\to\infty} ||w^{k+1} - \hat{w}^*||^2 \leq ||w^{k_j} - \hat{w}^*||^2 - \sum_{i=k_j}^{\infty} \frac{(L(w^i) - L^*)^2}{||g^i||^2}. \tag{4.56}$$

When $j \to \infty$ and using $||w^{k_j} - \hat{w}^*|| \to 0$ and (4.50), we get

$$\lim_{k\to\infty} ||w^{k+1} - \hat{w}^*||^2 \leq \lim_{j\to\infty} \left(||w^{k_j} - \hat{w}^*||^2 - \sum_{i=k_j}^{\infty} \frac{(L(w^i) - L^*)^2}{||g^i||^2} \right) = 0 \tag{4.57}$$

which implies that the whole sequence converges to $\hat{w}^* \in S^*$ or (4.46) holds. □

Note that the result from the Lemma above also holds when $\alpha_k \leq \frac{L(w^k)-\overline{L}}{||g^k||^2}$. In fact one can notice that Proposition 2.2.5 holds when the step size of the ordinary subgradient is chosen to be the modified Polyak step size.

After we proved convergence of subgradient method with modified Polyak step size when the optimal function value is replaced with an upper bound, we want to prove the same result for the deflected subgradient method. To this end we prove that for a certain step size the deflected subgradient is in fact a subgradient direction. We prove this result in the following two Lemmas which we adapt from [Gut03, Theorem 3.10 and Theorem 3.11].

Lemma 4.3.2. ***[Deflected subgradient direction is a subgradient direction in sense of*** (4.40)*]*** *Let g^k be a subgradient of $L(w^k)$ and f^k the deflected subgradient given by* (4.44) *with β^k as chosen in line (7) in the Algorithm 3. Let us denote with L^* the optimal value of the objective L. Let $\{w^k\}_k$ be a sequence of iterates in the deflected subgradient optimization. If for the step size α_k the following holds*

$$0 \leq \alpha_k \leq \frac{L(w^k) - L^*}{||f^k||^2} \tag{4.58}$$

then

$$\langle f^k, w^k - w^* \rangle \geq \langle g^k, w^k - w^* \rangle, \quad \forall k. \tag{4.59}$$

Proof. By induction on k. For $k = 0$, and as in the input in Algorithm 3 we have $f^k = 0$ and $g^k = 0$, in which case equality holds. Lets assume the inequality (4.44) holds for k, that is $\langle f^k, w^k - w^* \rangle \geq \langle g^k, w^k - w^* \rangle$. We have to prove it holds for $k + 1$. From (4.44) we have

$$\langle f^{k+1}, w^{k+1} - w^* \rangle = \langle g^{k+1}, w^{k+1} - w^* \rangle + \beta^{k+1} \langle f^k, w^{k+1} - w^* \rangle, \tag{4.60}$$

and from the condition on β^{k+1}, see line (7) in Algorithm 3, $\beta^{k+1} \geq 0$. In order to prove $\langle f^k, w^{k+1} - w^* \rangle \geq 0$, we use $w^{k+1} = P_S(w^k - \alpha_k f^k)$ and denote $p^{k+1} = P_S(w^k - \alpha_k f^k) - (w^k - \alpha_k f^k)$:

$$\begin{aligned} \langle f^k, w^{k+1} - w^* \rangle &= \langle f^k, p^{k+1} + w^k - \alpha_k f^k - w^* \rangle && (4.61) \\ &= \langle f^k, w^k - w^* \rangle - \alpha_k ||f^k||^2 + \langle f^k, p^{k+1} \rangle && (4.62) \\ &\geq \langle f^k, w^k - w^* \rangle - \alpha_k ||f^k||^2. && (4.63) \end{aligned}$$

where the last inequality follows from $\langle f^k, p^{k+1} \rangle \geq 0$ [Gut03, Lemmma 3.9]. This inequality follows from the properties of the projection onto a convex set which ensures that the angle between f^k and p^{k+1} is not obtuse. Hence, the cosine of the angle has to be non-negative. From the step size condition (4.58), g^k a subgradient of L, that is (4.40) and the assumption that (4.59) holds for k we have

$$0 \leq \alpha_k ||f^k||^2 \leq L(w^k) - L^* \leq \langle g^k, w^k - w^* \rangle \leq \langle f^k, w^k - w^* \rangle \tag{4.64}$$

from which follows

$$\langle f^k, w^{k+1} - w^* \rangle \geq \langle f^k, w^k - w^* \rangle - \alpha_k ||f^k||^2 \geq 0. \tag{4.65}$$

Consequently, the result holds for $k+1$

$$\langle f^{k+1}, w^{k+1} - w^* \rangle \geq \langle g^{k+1}, w^{k+1} - w^* \rangle. \tag{4.66}$$

□

With the Lemma above we proved that for certain choice of step size the deflected subgradient direction satisfies (4.40), i.e. it is a subgradient of L.

The next Lemma will show two important properties of the subgradient direction, extended to the deflected subgradient direction.

Lemma 4.3.3. ***[Properties of the deflected subgradient direction with a modified Polyak step size]*** *Let $\{w_k\}_k$ be a sequence of the iterates of the deflected subgradient method. Assuming the conditions in Lemma 4.3.2 are fulfilled and the step size is chosen as in (4.58), then for both non-optimal w^k and optimal w^* it holds:*

(a) $\langle f^k, w^k - w^* \rangle > 0$

(b) $||w^{k+1} - w^*|| < ||w^k - w^*||$

Proof. (a) Obvious due to the result of Lemma 4.3.2 and (4.40).

(b) We have

$$\begin{aligned} ||w^* - w^{k+1}||^2 &\leq ||w^* - w^k + \alpha_k f^k||^2 && (4.67a) \\ &= ||w^* - w^k||^2 + \alpha_k(\alpha_k ||f^k||^2 - 2\langle f^k, w^k - w^* \rangle) && (4.67b) \end{aligned}$$

where the first inequality follows from the non-expansivity property of a projection (in the general case of subgradient projection). From the step size condition (4.58)

we have for the term in the parenthesis

$$\alpha_k||f^k||^2 - 2\langle f^k, w^k - w^*\rangle \leq L(w^k) - L^* - 2\langle f^k, w^k - w^*\rangle \tag{4.68a}$$
$$< 2(L(w^k) - L^*) - 2\langle f^k, w^k - w^*\rangle \leq 0, \tag{4.68b}$$

where the last inequality follows from (4.40) and the result from the Lemma above (4.59). From here we have $||w^{k+1} - w^*||^2 < ||w^k - w^*||^2$ and so (b) holds. □

The following Lemma adopted from [Gut03, Theorem 3.12] shows a stronger property of the deflected subgradient direction compared to an ordinary subgradient method, given that a modified Polyak step size and certain choice of the parameter β in (4.44) is used.

Lemma 4.3.4. ***[Convergence of the deflected subgradient direction with a modified Polyak step size]*** *Let $\{w^k\}_k$ be a sequence of iterates in the deflected subgradient optimization. Let the sequence of step size α_k fulfills*

$$0 \leq \alpha_k \leq \frac{L(w^k) - L^*}{||f^k||^2}, \qquad k = 0, 1, .. \tag{4.69}$$

where L^* denotes the optimal value of L and f^k the deflected subgradient given by (4.44), with

$$\beta^k = \begin{cases} -\gamma_k \frac{g^k(f^{k-1})^\top}{||f^{k-1}||^2}, & \text{if } g^k(f^{k-1})^\top < 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.70}$$

with $0 \leq \gamma_k < 2$, and g^k the subgradient of $L(w^k)$. Then it holds

$$\frac{\langle f^k, w^k - w^*\rangle}{||f^k||} \geq \frac{\langle g^k, w^k - w^*\rangle}{||g^k||}. \tag{4.71}$$

Proof. If $g^k(f^{k-1})^\top \geq 0$, $\beta^k = 0$ then from (4.44) $f^k = g^k$, in which case (4.71) holds. If $g^k(f^{k-1})^\top < 0$, $\beta^k = -\gamma_k \frac{g^k(f^{k-1})^\top}{||f^{k-1}||^2}$ and

$$||f^k||^2 - ||g^k||^2 \overset{(4.44)}{=} ||g^k + \beta^k f^{k-1}||^2 - ||g^k||^2 \tag{4.72}$$
$$= \beta^k(2g^k(f^{k-1})^\top + \beta^k||f^{k-1}||^2) \tag{4.73}$$
$$\overset{(4.70)}{=} \beta^k(2g^k(f^{k-1})^\top - \gamma_k g^k(f^{k-1})^\top) \tag{4.74}$$
$$= \beta^k(2 - \gamma^k)g^k(f^{k-1})^\top \tag{4.75}$$
$$\leq 0 \tag{4.76}$$

where the last result comes from the condition $0 \leq \gamma^k < 2$, $g^k(f^{k-1})^\top < 0$ and $\beta^k \geq 0$. As a result we have $||f^k||^2 \leq ||g^k||^2$. Using this result and Lemma 4.3.2 we finally get

$$\frac{\langle f^k, w^k - w^*\rangle}{||f^k||} \geq \frac{\langle g^k, w^k - w^*\rangle}{||g^k||}. \tag{4.77}$$

□

This result above (4.77) also implies that the angle between the negative deflected subgradient direction to the optimal set is smaller than the angle between the negative deflected subgradient direction, which means faster convergence. It can happen that the ordinary subgradient direction forms an obtuse angle (angle bigger than 90^o) with the previous deflected direction which implies $\langle f^{k-1}, g^k \rangle < 0$. This would result in a zigzag behavior of the iterates. In this case the subgradient direction is ***corrected*** to be the deflected direction f^k. This correction is done with a suitable choice of the parameter β in every iteration. For the deflected direction, however, $\langle f^{k-1}, f^k \rangle \geq 0$ is always satisfied when the parameter γ in β (4.70) is chosen to be $\gamma \geq 1$, see [Gut03, Theorem 3.13]. As a consequence, the zigzagging occurring in the subgradient direction and slowing down convergences is ***curred*** by the deflected subgradient direction. We now state the main convergence result for our Algorithm 3.

Theorem 4.3.5. ***[Convergence of Algorithm 3]*** *Let g^k be a subgradient of $L(w^k)$ and f^k the deflected subgradient given by (4.44) with β^k as chosen in line (7) in the Algorithm 3. If for the step size α_k (4.69) holds, where L^* denotes the optimal value of L for a sequence $\{w^k\}_k$ of iterates in the deflected subgradient optimization, then $w^k \underset{k\to\infty}{\to} w^*$, where $w^* \in S^*$, is an optimal value.*

Proof. Since we don't know the optimal value L^* in the step size α_k as in line (11) in Algorithm 3, we use an approximation of the optimal value, $\overline{L}$. If $\overline{L}$ is an upper bound on L^*, then

$$0 \leq \alpha_k = \frac{L(w^k) - \overline{L}}{||f^k||^2} \leq \frac{L(w^k) - L^*}{||f^k||^2}. \tag{4.78}$$

If it happens that $\overline{L}$ is a lower bound on L^* we increase $\overline{L}$ in line (15-16) of Algorithm 3 every certain number of iterations. Eventually, after a finite number of iterations we will get a value for $\overline{L}$ such that $\overline{L} > L_{min} - \epsilon_1$, where L_{min} is the minimum value of L, ϵ_1 is some small threshold value. Note that $\epsilon_1 > \epsilon_2$ must hold, with ϵ_2 defining the stopping condition. For our implementation we choose $\epsilon_1 = 10^{-5}$. From here $\overline{L} \geq L^*$. This implies (4.78) and so the conditions of Lemma 4.3.4 are fulfilled. As a result after a finite number of iterations (4.59) and (4.71) hold. From (4.59) and (4.40) the deflected subgradient f is a subgradient, too, so all the results that hold for the subgradient will hold for the deflected subgradient as well. This implies that the convergence result for the Polyak step size will also hold for the deflected subgradient method when the optimal function value is replaced by an upper bound. Then from Lemma 4.3.1 is follows that $w^k \underset{k\to\infty}{\to} w^*$, that is the deflected subgradient with modified Polyak step size as implemented in Algorithm 3 converges to an optimal solution. □

Remark 4.3.4. Algorithm 3 might converge slowly, e.g. due to selecting a very small upper bound, like in our input $\overline{L} = 0.1$. This can be a lower bound and will take a lot of iterations to be adjusted to the actual upper bound. However, we can always choose different input parameters, that is bigger $\overline{L}$ and smaller n, from the

input line for Algorithm 3. If not stated otherwise, for all experiments we will use the parameters as described in Algorithm 3.

Remark 4.3.5. For the parameter γ_k in (4.70), we use a constant value $\gamma_k = 1.5$, $\forall k$ as suggested in [CFM75].

4.3.5. Comparison of the Linearized Approach to Structured SVM

Structured SVM [FJ08, TJHA05, THJA04] generalizes the classification SVM to general structured output labels and is defined by

$$\min_{w,\xi} \frac{1}{2}||w||^2 + C\xi, \quad \text{s.t. } \xi \geq 0 \tag{4.79a}$$

$$\frac{1}{N}\sum_{k\in[N]} \langle \theta_w, \mu - \mu_k^* \rangle \geq \frac{1}{N}\sum_{k\in[N]} \triangle(\mu, \mu_k^*) - \xi \tag{4.79b}$$

where θ_w are the model parameters depending on the vector w which is to be learned, ξ is a slack variable which is added in order to allow errors in the training data and C is a parameter that controls the weighting between the training error and the maximum margin of separation. N is the number of training instances, $\triangle(\mu, \mu_k^*)$ denotes a loss function which can be designed depending on the type of the problem, with μ_k^* being ground truth labeling of the training data k and a given training labeling μ such that $\mu \neq \mu_k^*$, $k \in [N]$. Structured SVM tries to find a minimal ℓ_2-norm w, so that the training error i.e. the error between the obtained segmentation to the ground truth observed segmentation regulated through ξ is as small as possible, while uniqueness is enforced by choosing w for which the margin of separation of the predicted and the ground truth score is the biggest.

In our linearized approach (LA) we can optionally add the norm of w in the objective to make sure there is no overfitting, see Remark 4.3.1. The uniqueness in our approach was enforced by the ε term (4.30). We minimize the function L (our loss function) in (4.38) which we rewrite as

$$\min_{w} \sum_{k\in[N]} \left(\langle \tilde{\theta}_w, \mu_k^* \rangle - \max_{\mu\in L_{\mathcal{G}}} \langle \tilde{\theta}_w, \mu \rangle\right). \tag{4.80}$$

A loss function similar to ours, $\sum_{k\in[N]} \left(\langle \tilde{\theta}_w, \mu_k^* \rangle - \max_{\mu\in L_{\mathcal{G}}} \langle \tilde{\theta}_w, \mu \rangle\right)$, was used in [LH05] and referred to as the generalized perceptron loss function. However, the authors consider a minimization instead of a maximization problem.

Now let us substitute our loss function (4.80) into the structured SVM problem. Adding $1/N$ before the loss function does not affect the final result, and we get:

$$\min_{w,\xi} \frac{1}{2}||w||^2 + C\xi, \quad \text{s.t. } \xi \geq 0 \tag{4.81a}$$

$$\frac{1}{N}\sum_{k\in[N]} \langle \theta_w, \mu - \mu_k^* \rangle \geq \frac{1}{N}\sum_{k\in[N]} \left(\langle \tilde{\theta}_w, \mu_k^* \rangle - \max_{\mu\in L_{\mathcal{G}}} \langle \tilde{\theta}_w, \mu \rangle\right) - \xi. \tag{4.81b}$$

Taking into account the reparametrized potentials $\tilde{\theta}$ as defined in (4.35) we have

$\tilde{\theta}(x^*) = \theta$, while $\tilde{\theta}(x) = \theta - \varepsilon$ for $x \neq x^*$, for all nodes and edges different than the ground truth segmentation. Using this result, we can reformulate the problem as

$$\min_{w,\xi} \frac{1}{2}||w||^2 + C\xi, \quad \text{s.t. } \xi \geq 0 \tag{4.82a}$$

$$\frac{1}{N}\Big(\sum_{k\in[N]} \langle \theta_w, \mu - 2\mu_k^* \rangle + \max_{\mu\in L_{\mathcal{G}}} \langle \theta_w, \mu \rangle - \max_{\mu\in L_{\mathcal{G}}} \langle \varepsilon, \mu \rangle \Big) + \xi \geq 0. \tag{4.82b}$$

If we set $\max_{\mu\in L_{\mathcal{G}}} \langle \varepsilon, \mu \rangle = \xi(N-1)$, it becomes clear that the purpose of the hand-chosen ε is similar to the slack variable ξ.

In our objective (4.80) we are minimizing over one variable which is clearly an easier to optimize function, as opposed to the objective in structured SVM where there are two variables to be optimized over. Our optimization procedure as described in Sect. 4.3.3 differs from the ones proposed for the structured SVM. Despite using sophisticated subgradient optimization our method is less complex than cutting plane methods [JFY09], the most efficient ones for structured SVMs.

We can conclude that our objective which is directly minimizing the loss function as in (4.80) closely resembles an objective with structured SVM learning, see (4.81) where we inserted the same loss function as in our objective. However our objective comes down to a less complex optimization when compared to structured SVM. While we minimize a loss function over one variable in our objective, we enforce uniqueness too, with setting an additional term by hand.

4.4. Difference Between the Two Approaches

The two approaches presented here both solve the problem of learning parametrized potentials θ for the relaxed binary labeling problem (4.3). However, the two-step invLPA, see Sect. 4.2, and the linearly parametrized approach (LA), see Sect. 4.3 , take two completely different approaches to solve the problem.

With invLPA following the method for inverse linear optimization as in [ZL96, AO01] we exploit the KKT conditions in order to come up with the inverse linear program which finds exact perturbations, whereas with LA we exploit the dual in order to come up with the convex relaxation of the original non-convex bi-level optimization problem. In order to gain a tight convex relaxation when using LA and being able to use off-the-shelf inference solvers, it is essential to build on linear dependencies of the parametrized potentials with respect to the w vectors being learned. In contrast invLPA is not restricted to any potential model, which is major advantage since we are very flexible to use any model parameter prediction method.

The invLPA method separates learning in two separate and independent steps: the first computes the exact perturbations (corrected potentials), with linear optimization, while the second step fits the learned potentials to given data features. And advantage of LA in this respect is that it solves both the perturbation and fitting problem jointly in one step, but with a more involved optimization method, an improved subgradient procedure for solving the parametrized subproblems of the saddle point problem, (4.38).

While our LA approach resembles structured SVM learning methods, our invLPA is a completely novel method as to our knowledge. It is the first method which uses inverse linear programming [ZL96, AO01] for learning parameters in graphical models. And the first method which can find exact perturbations corresponding to given ground truth data, that is model parameters which lead to given exact ground truth data. This, however, applies to the training data.

Concerning empirical comparison of the two presented methods we refer to the next Sect. 4.5.

4.5. Experiments and Discussion

In this section we evaluate:

1. The first approach, invLPA from Section 4.2 which computes for each training image a model parameter perturbation that exactly has the corresponding ground truth labeling as global minimum, followed by fitting a model parameter vector to these perturbations.
2. The second approach, LA from Section 4.3 which solves the same problem in one step, or jointly computes perturbations and fits a model parameter vector.

We perform experiments and do the following validations and comparisons:

1. We validate the ability of invLPA to find the exact perturbed model parameters by performing an experiment with ground truth data designed for our purpose.
2. We validate the performance of invLPA, as well as the performance of LA, by
   a) learning unary potentials, considering presence of a regularizer, an Ising prior. This is a different objective than the standard training of a classifier offline followed by plugging in the learned unary potentials into an energy function which contains a regularizer.
   b) Learning pairwise potentials, with very weak unary terms for a scenario for which any standard regularizer, for instance Ising prior would certainly fail.
3. We validate the performance of the two approaches on the Weizmann horse dataset [BU08] by:
   a) learning jointly unary and pairwise terms using LA with comparison to training a linear SVM on the unary data,
   b) learning pairwise terms using invLPA and
   c) comparison to the learning method from [Dom13], with two different loss functions, one based on Maximum a Posteriori Marginal (MPM) and one based on Maximum Likelihood Estimate (MLE).
4. We compare the two approaches, invLPA and LA with respect to unary potential learning as well as pairwise potential learning. In addition we explain the drawbacks for both learning methods, invLPA and LA.

4.5.1. Ground Truth Experimental Evaluation of invLPA

We want to demonstrate the ability of our invLPA approach to learn correct model parameters which lead to ground truth segmentations, see (4.9). We will see from the experiments in the following sections that our invLPA approach always finds some corrected potentials that lead to the exact ground truth segmentation. There is indeed a set of model parameters which lead to the desired segmentation, see (4.9). But what happens if we know unique ground truth model parameters, will our invLPA method lead to these ground truth potentials? Or how will the potentials obtained from invLPA differ to the ground truth ones? In order to give answers to these questions we perform a simple experiment with data we generated. We fix probability distributions and sample from it a simple chain graph with 128 nodes. From the probability distribution we can compute the model parameters. In the following we explain in details what we exactly do.

Sampling We use a chain graph which is an acyclic graph and a tree. This allows to apply the theory of probability distributions on trees. Drawing samples from a tree distribution is based on sampling from local conditional distributions. Given a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ a tree distribution $p(x)$ can be represented as in (2.95). Taking into account that we have an acyclic graph, the marginals are the corresponding probabilities, we rewrite the factorized probability distribution as

$$p(x) = \frac{\prod_{ij \in \mathcal{E}} p(x_i, x_j)}{\prod_{i \in \mathcal{V}} p(x_i)^{d(i)-1}}, \tag{4.83}$$

where $d(i) = |\mathcal{N}(i)|$ is the degree of a node i, for $\mathcal{N}(i)$ denoting the set of adjacent nodes to i. We consider the tree $\mathcal{T}$ as a directed chain with $\mathcal{V} = \{0, 1, \ldots, n\}$. Then, since $d(0) = d(n) = 1$ and $d(i) = 2$ if $1 \leq i \leq n-1$, (4.83) reads

$$p(x) = \frac{p(x_n, x_{n-1})}{p(x_{n-1})} \cdot \ \cdots \ \cdot \frac{p(x_1, x_0)}{p(x_0)} p(x_0) \tag{4.84a}$$

$$= p(x_n|x_{n-1}) \cdot \ \cdots \ \cdot p(x_1|x_0) p(x_0). \tag{4.84b}$$

Thus, we can sample from the probability distribution, that is $x \sim p(x)$ can be computed sequentially by $x_0 \sim p(x_0)$, $x_1 \sim p(x_1|x_0), \ldots, x_n \sim p(x_n|x_{n-1})$.

We can compute the model parameters from the binary probability distribution in the following way. We first consider the marginal representation (4.84) of a binary

distribution on a chain graph and compute

$$\begin{aligned}
\log p(x) &= \sum_{i\in[n]} \log p(x_i, x_{i-1}) - \sum_{i\in[n-1]} \log p(x_i) && \text{(4.85a)}\\
&= \sum_{i\in[n]} \Big(\log p(x_i = 0, x_{i-1} = 0)(1 - x_i)(1 - x_{i-1}) && \text{(4.85b)}\\
&\quad + \log p(x_i = 0, x_{i-1} = 1)(1 - x_i)x_{i-1} && \text{(4.85c)}\\
&\quad + \log p(x_i = 1, x_{i-1} = 0)x_i(1 - x_{i-1}) + \log p(x_i = 1, x_{i-1} = 1)x_i x_{i-1}\Big) && \text{(4.85d)}\\
&\quad - \Big(\log p(x_{n-1} = 0)(1 - x_{n-1}) + \log p(x_{n-1} = 1)x_{n-1} && \text{(4.85e)}\\
&\quad + \log p(x_1 = 0)(1 - x_1) + \log p(x_1 = 1)x_1\Big) && \text{(4.85f)}\\
&= \eta_{n(n-1)}(0, 0)(1 - x_n)(1 - x_{n-1}) + \cdots - \eta_1(1)x_1 && \text{(4.85g)}\\
&= \sum_{i\in[n]} \theta^{i(i-1)} x_i x_{i-1} + \sum_{i\in[n-1]} \theta^i x_i. && \text{(4.85h)}
\end{aligned}$$

We note that the canonical parameters above are in a minimal representation form. The last equality in the equation above is due to the exponential model representation, see (2.49), which we write here again without the log-partition function, as no normalizing factor is required, since the sum of both sides over all binary vectors x is equal to 1. This follows from the fact that we used the ***marginal representation*** of $p(x)$ which is only possible on an acyclic graph. We write the exponential model as

$$p(x) = \exp\big(\langle\theta, \phi(x)\rangle\big), \qquad \phi(x) = (x_n x_{n-1}, \ldots, x_1 x_0, x_{n-1}, \ldots, x_1). \tag{4.86}$$

We fix the values of the local marginal distributions as

$$\begin{aligned}
p(x_{i-1} = 0) &= \alpha && \text{(4.87a)}\\
p(x_{i-1} = 1) &= 1 - \alpha && \text{(4.87b)}\\
p(x_{i-1}, x_i) &= p(x_i|x_{i-1})p(x_{i-1}) && \text{(4.87c)}
\end{aligned}$$

where the pairwise distribution is computed by fixing the conditional distribution to

$$p(x_i|x_{i-1}) = \begin{pmatrix} \beta & 1-\beta \\ \gamma & 1-\gamma \end{pmatrix} \tag{4.88}$$

where $i = 1, ..., n$, with $n+1$ the size of the chain graph (n nodes plus the root node), 129 in our case. We write the conditional probability in the form (4.88) due to the ancestral sampling which we use and requires this form of conditional probabilities. We take $\alpha = 0.7$, $\beta = 0.9$ and $\gamma = 0.2$. This fixes both η and θ by the preceding

relations. That is, from (4.85) the canonical parameters θ can be expressed as

$$\begin{aligned}
\theta^0 =& \eta_0(0) - \eta_0(1) - \eta_{1,0}(0,0) + \eta_{0,1}(0,1) && \text{(4.89a)}\\
\theta^{i-1} =& \eta_{i-1}(0) - \eta_{i-1}(1) - \eta_{i,i-1}(0,0) + \eta_{i,i-1}(0,1) - \eta_{i-1,i-2}(0,0) && \text{(4.89b)}\\
&+ \eta_{i-1,i-2}(1,0) \quad \text{for } i = 2,..n && \text{(4.89c)}\\
\theta^{i,i-1} =& \eta_{i,i-1}(0,0) - \eta_{i,i-1}(0,1) - \eta_{i,i-1}(1,0) + \eta_{i,i-1}(1,1) \quad \text{for } i = 1,....n, && \text{(4.89d)}
\end{aligned}$$

where the parameters η_{ij}, η_i are the log-values of the local marginals of the chain distribution

$$\begin{aligned}
\eta_{i-1}(0) &= \log(p(x_{i-1} = 0)) && \text{(4.90a)}\\
\eta_{i-1}(1) &= \log(p(x_{i-1} = 1)) && \text{(4.90b)}\\
\eta_{i,i-1}(0,0) &= \log(p(x_{i-1} = 0, x_i = 0)) && \text{(4.90c)}\\
\eta_{i,i-1}(0,1) &= \log(p(x_{i-1} = 0, x_i = 1)) && \text{(4.90d)}\\
\eta_{i,i-1}(1,0) &= \log(p(x_{i-1} = 1, x_i = 0)) && \text{(4.90e)}\\
\eta_{i,i-1}(1,1) &= \log(p(x_{i-1} = 1, x_i = 1)) && \text{(4.90f)}\\
& && \text{(4.90g)}
\end{aligned}$$

for $i = 1, ..., n$.

We draw 1000 samples. After having generated our data and computed ground truth model parameters we want to perform the perturbation step of our invLPA method. We use a simple scenario where we learn only the data term and consider the pairwise terms as fixed, the ground truth ones. In oder to apply invLPA we need some initial model parameters. For this we use noisy version of our data.

Remark 4.5.1. In what follows, we use the vertex set $\mathcal{V} = [n] = \{1, \ldots, n\}$. The node 0 was only used above to highlight a particular node as the "root" of an oriented chain, to explain the sampling procedure.

Data Term and Inference In practice we do not observe x directly, but the data vector $y = \{y_1, \ldots, y_n\}$, which can be binary or real. A data observation model taking into account noise has to be specified in terms of the conditional data likelihood distribution

$$p(y|x) = \prod_{i \in [n]} p(y_i|x_i), \tag{4.91}$$

that we adopt in the simplest possible form: fully factorized which means conditional independence of the data given x, and functional form that is independent of the location i (i.e. the same distribution p on the right at every vertex i). Then we choose a prior $p(x)$ which typically is just the 'edge part' (smoothness prior) above corresponding to $\phi(x) = (x_1 x_2, \ldots, x_{n-1} x_n)$. MAP inference then means

$$\arg\max_x p(x|y) = \arg\max_x p(x|y)p(y) = \arg\max_x p(y|x)p(x), \tag{4.92}$$

because the term $p(y)$ does not depend on x. It remains to rewrite (4.91) in exponential form in order to conduct inference with the LP relaxation.

We consider the case of binary data vectors y and write

$$l_{00} = \log p_{00} = \log p(y_i = 0 | x_i = 0) \tag{4.93}$$

etc. Then

$$\log p(y|x) = \sum_{i \in [n]} \log p(y_i|x_i) \tag{4.94a}$$

$$\log p(y_i|x_i) = (l_{00}(1-y_i) + l_{10}y_i)(1-x_i) + (l_{01}(1-y_i) + l_{11}y_i)x_i \tag{4.94b}$$

$$= ((l_{00} + l_{11} - l_{10} - l_{01})y_i + l_{10} - l_{00})x_i + (\text{terms independent of } x_i) \tag{4.94c}$$

$$= \theta_i(y_i)x_i. \tag{4.94d}$$

Absorbing terms that do not depend on x into the normalizing constant, we get

$$p(y|x) = \exp\Big(\sum_{i \in [n]} \log p(y_i|x_i)\Big) \propto \exp(\langle \theta(y), x \rangle), \quad \theta(y) = \big(\theta^1(y_1), \dots, \theta^n(y_n)\big)^\top. \tag{4.95}$$

An elementary data likelihood model with noise level ε is

$$p(y_i|x_i) = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} = \begin{pmatrix} 1-\varepsilon & \varepsilon \\ \varepsilon & 1-\varepsilon \end{pmatrix}, \qquad \varepsilon \in (0,1), \tag{4.96}$$

for all i from the training data. This results in the canonical parameters

$$\theta^i(y_i) = 2\log\Big(\frac{1-\varepsilon}{\varepsilon}\Big)y_i - \log\Big(\frac{1-\varepsilon}{\varepsilon}\Big) = \begin{cases} -\log\big(\frac{1-\varepsilon}{\varepsilon}\big) & \text{if } y_i = 0 \\ +\log\big(\frac{1-\varepsilon}{\varepsilon}\big) & \text{if } y_i = 1 \end{cases} \qquad i \in \mathcal{V}. \tag{4.97}$$

To motivate the above choice of data term, we can consider local decisions solely based on this likelihood model. If $\varepsilon \searrow 0$ then $\log\big(\frac{1-\varepsilon}{\varepsilon}\big) \to +\infty$, which implies the decision $x_i = y_i$ in order to maximize the energy $\theta^i(y_i)x_i$, which makes sense in the case of no noise $\varepsilon \approx 0$. On the other hand, for increasing noise levels this data term becomes weaker, and if $\varepsilon > 0.5$ it even changes the sign, because then $x_i \neq y_i$ is more likely the correct decision.

Perturbation We sample observed values $y \in \{0,1\}$ from the samples x with a fixed conditional probability as in (4.96). We take three different choices of noise $\varepsilon_1 = 0.01$, $\varepsilon_2 = 0.1$ and $\varepsilon_3 = 0.6$.

In the first step of the inverse LP approach we find correction to initial $\hat{\theta}$ parameters on the data term only, while fixing the smoothness prior to the computed ground truth potentials from the probability distribution we sample from.

We illustrate for one sample and the three different noise levels ε for the initial $\hat{\theta}$ as in (4.97) in Fig. 4.3 the data term we use, the corrected potentials on the data

term and the correct ground truth unary parameters corresponding to the probability distribution from which we sample.

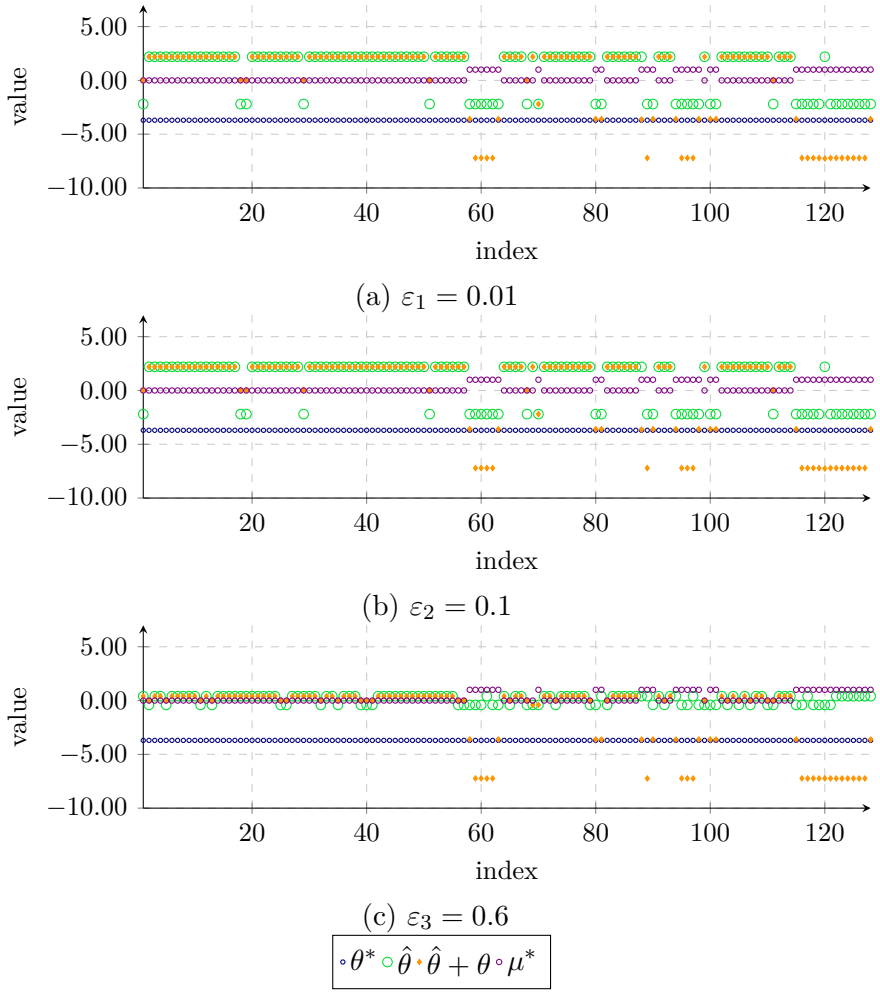
For all noise levels the corrected potentials lead to ground truth segmentation. However, as we can see from Fig. 4.3 the initial potentials are only corrected in those nodes where $\mu^* = 1$. Moreover the corrected potentials $\theta + \hat{\theta}$ correspond to the ground truth potentials which we calculated from our sampling distribution at the end nodes of the 1 segments. First the set of optimal θ which lead to optimal μ^* is not unique, i.e. the set (4.9) contains more than one element. This is part of the explanation why the corrected potentials which do not match the ground truth ones still lead to the ground truth segmentation. The reason why for the rest of the 1 segments we have $\hat{\theta} + \theta < \theta^*$ is the minimization problem $\min_{\mu \in \mathcal{L}_\mathcal{G}} \langle \mu, \theta + \hat{\theta} \rangle$, which will get a minimum for smaller θ values corresponding to 1 nodes, while $\theta + \hat{\theta}$ leads to the same ground truth μ^*, see (4.9). This is as well the second part of the explanation why the corrected potentials match the ground truth segmentation. As for the nodes where $\mu^* = 0$ there is no correction, except in few cases when the initial $\hat{\theta} < 0$, then the potentials get a correction, such that $\hat{\theta} + \theta = 0$. This is the case if both $\nu = 0$ and $\nu_\mu = 0$ as discussed in (4.18).

As a conclusion we can say that our invLPA method always corrects the initial potentials so that the corrected ones lead to the exact ground truth segmentation. However, this is achieved only by correcting the nodes corresponding to the foreground segment. As a result the predicted potentials will depend on the initial potentials too. When we have some initial $\hat{\theta}$ obtained with some other learning method we can always expect to get an improvement as our invLPA method will correct some of the potentials corresponding to foreground nodes. The background nodes will, however, always stay as they are. The fitting in the next step will then depend on the corrected potentials. In other words we can rely on our invLPA method to give improved predicted potentials only when we have initial potentials obtained from another learning method. Moreover, we only need initial potentials for the 0 segments.

The same argument follows for the pairwise potentials. invLPA will correct only those pairwise potentials that correspond to edges which connect nodes that both belong to a foreground segment.

4.5.2. Learning Unary Potentials

In this section we evaluate the two approaches for learning unary terms in a presence of a fixed regularizer. We want to demonstrate that learning the unary potentials while considering an objective where a regularizer is present, too, has a benefit over a learning method where the unary potentials are learned offline and a regularizer is added afterwards. Moreover, we want to demonstrate the benefit of our invLPA method which learns corrected initial potentials.



(a) $\varepsilon_1 = 0.01$

(b) $\varepsilon_2 = 0.1$

(c) $\varepsilon_3 = 0.6$

Figure 4.3. - Initializing with three different noise levels. In all three cases invLPA finds the correct ground truth potentials only at the end points of each of the 1 segments ($\mu^* = 1$). For the 1 segments when $\hat{\theta} + \theta < \theta^*$ is due to the minimization problem $\min_{\mu \in \mathcal{L}_\mathcal{G}} \langle \mu, \theta + \hat{\theta} \rangle$, which will get a minimum for smaller θ values corresponding to 1 nodes and a set of potentials $\theta + \hat{\theta}$ leading to the same ground truth μ^*, see (4.9). As for the 0 segments ($\mu^* = 0$) the potentials are not corrected (except few which have negative $\hat{\theta}$ are corrected to result in $\hat{\theta} + \theta = 0$). The reason that for the 0 segments the potentials are not well corrected is due to the form of the matrix A in (4.18). We are searching for sparse correction θ to $\hat{\theta}$ such that the conditions from (4.18) are satisfied. This is equivalent to finding sparse θ such that $\mu^* = \arg\min_{\mu \in \mathcal{L}_\mathcal{G}} \langle \mu, \theta + \hat{\theta} \rangle$. When $\mu = 0$, $\hat{\theta} + \theta$ can be anything and this will not change the optimal solution, which is why $\theta = 0$ in this case. This is also the reason why the corrected potentials still lead to the ground truth segmentation μ^*.

Experimental Set-Up

For validation of the two approaches for learning unary potentials we used 100 generated images with random lines, vertical and horizontal, see Fig. 4.4. All the images are the same size 64×64 pixels. For features we used correlated images with 8 image filters as illustrated in Fig. 4.5. After correlating an input image, the resulting values were encoded for each pixel by stacking corresponding unit-vectors. This resulted in feature vectors $f^i \in \mathbb{R}^{97}$ with a large number of degrees of freedom for learning. In order to exclude errors on the border due to the correlation we ignore the 4 closest pixels to the border from the images of feature values. Accordingly, we will work with images of dimension only 56×56. For the invLPA approach we wish to start with a strong data term $\hat{\theta}_u = (..., \langle f^i, w_u \rangle, ...)$ in order to investigate if it can further improve the result. LA on the other hand is not sensitive to initialization. To this end we trained a linear classifier using logistic regression and 10 independent foreground and 10 independent background images, see Fig. 4.4. The training error on these images was: 99 wrong foreground pixels and 140 wrong background pixels, out of $62720 = 56 \times 56 \times 10 \times 2$ or 0.38% mislabeled pixels in total.

We note that from these 10 foreground and 10 background images we generate our 100 images which we divide into 70% train and 30% test dataset.

In the experiments to follow, as an error measure we use the percentage of mislabeled pixels similarly like in [BYVG11]. We calculate the percentage of mislabeled pixels with

$$100 \frac{\sum_p (I(p) \neq I_{gt}(p))}{|p|} \tag{4.98}$$

where I is the final segmented image we learn, I_{gt} is the ground truth segmentation and $|p|$ is the total number of pixels in the image.

Applying invLPA

Next we explain the implementation details for invLPA.

This approach consists of two independent steps: perturbation and fitting which we separately explain in the following.

Perturbation We consider N training samples. For simplicity of notation we consider N to be the set of all nodes through all the images, although this step can be parallelized, when we consider each image separately and compute perturbations in parallel for each image. We have for each image its ground truth segmentation, where μ_i^*, $i \in [N]$, μ_i^* here denotes the ground truth label of the node i. We compute perturbations θ^i to the initial potentials $\hat{\theta}^i$, $i \in [N]$ obtained with training state-of-the-art logistic classifier with the set up as explained in Fig. 4.4.

As stated before in section 4.2 for the invLPA we use the minimal representation for the local polytope relaxation. We consider μ^* to be the solution to (4.3) with $\mathcal{L}_\mathcal{G} = \mathcal{L}_\mathcal{G}^M$ given by (4.7) and $\hat{\theta}$ vector of model parameters such that $\mu^* \notin \arg\min_{\mu \in \mathcal{L}_\mathcal{G}^M} \langle \hat{\theta}, \mu \rangle$.

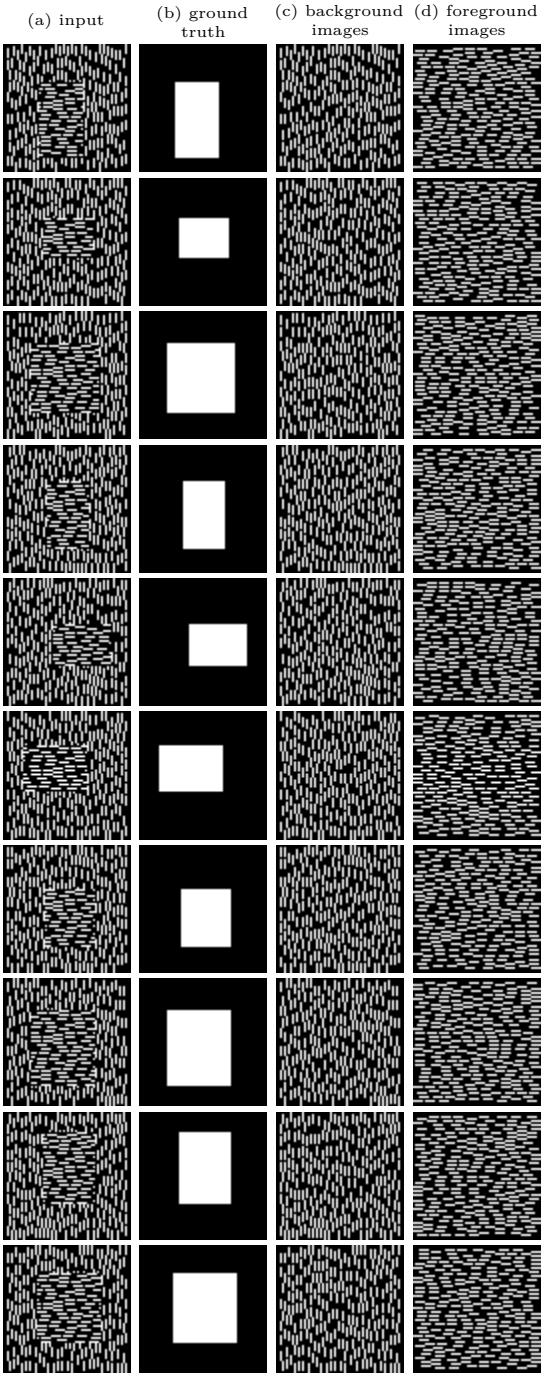


Figure 4.4. - (a) 10 images composed of two random processes with vertical and horizontal lines out of 100 along with (b) ground truth segmentations. (c) 10 independent background images and (d) 10 independent foreground images used for training a linear classifier using logistic regression in order to define a strong data term $\hat{\theta}$. Features were computed by using the templates depicted by Fig. 4.5.

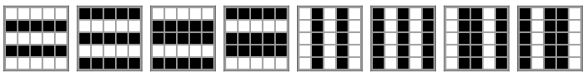


Figure 4.5. - 8 binary filter masks for computing the feature vectors corresponding to each pixel. After correlating an input image, the resulting values were encoded for each pixel by stacking corresponding unit-vectors. This resulted in feature vectors $f^i \in \mathbb{R}^{97}$ with a large number of degrees of freedom for learning. In view of the images from Fig. 4.4.

μ^* is represented as

$$\mu^* := (..., \mu_i^*, ..., \mu_{ij}^*, ...), \quad i \in \mathcal{V}, ij \in \mathcal{E} \tag{4.99}$$

where $\mu_i^* = 0$ if pixel i is in the background and $\mu_i^* = 1$ if pixel i is in the foreground, $\mu_{ij}^* = 1$ if both pixels i and j are in the foreground and 0 for all other cases. With $\mathcal{V}$ we denote the set of all nodes from the training data, $\mathcal{E}$ the set of all edges from the training data. For the unary case (learning only unary potentials) we considered a fixed Ising prior with weight β for the pairwise potentials. Setting $\beta = 2$ showed to be a good choice in this case. Then the initial potentials are defined as:

$$\hat{\theta} := (\hat{\theta}_u, \hat{\theta}_p) = (..., \langle \hat{w}_u, f^i \rangle + |\mathcal{N}(i)|\beta, ..., -2\beta, ...)^\top \in \mathbb{R}^{N+M}, i \in \mathcal{V} \tag{4.100}$$

where with $|\mathcal{N}(i)|$ we denote the number of neighbors of a node $i \in \mathcal{V}$, N is the number of nodes and M is the number of edges. With the subscript u we denote the unary part of the potentials and with the subscript p the pairwise part of the potentials. In the general case $\hat{w}_u$ is the initial guess for the parametrization $\hat{\theta}_u = \hat{\theta}_u(\hat{w}_u)$. In our case we obtained $\hat{w}_u$ by training a linear classifier using a logistic regression, and f^i is the feature vector corresponding to node $i \in \mathcal{V}$.

We compute the perturbations θ^i, $i \in [N]$ by solving the linear system (4.18).

Fitting In the second step, we do the fitting on the computed perturbed potentials, which now correspond to the exact ground truth labellings of the training data. We are given the features

$$f^i, \quad i \in [N] \tag{4.101}$$

and the corrected potentials

$$\tilde{\theta}_u^i := \langle \hat{w}_u, f^i \rangle + \theta_u^i = \text{initial potential} + \text{perturbation (correction)} \tag{4.102}$$

collected from all locations and from all training images. We can learn potentials $\tilde{\theta}$ employing any model prediction method. For the following experiments we used the three prediction methods described in Sect. 4.2.2.

1. **Linear fitting:** We used least-squares fitting with solving the overconstrained system (4.20) and ℓ_1-norm fitting, with solving (4.21).

2. **Nonlinear fitting:** We applied Gaussian regression in order to investigate if it will capture more from the structure than the simple linear methods.

method	In(LOG)	LS	L1	NL	LA
% mis	2.44	2.42	2.44	**2.00**	2.02

Table 4.1. - The numbers correspond to the mean errors from Fig. 4.7 and Fig. 4.8. The nonlinear Gaussian fitting, (NL) in average outperforms all other methods.

> For the nonlinear Gaussian fitting we used a sparse Gaussian regression with only 5% of the whole training data, chosen by random. We use the publicly available code from [RW06] for optimizing the regression parameters.

In Fig. 4.7 we compare the error of the investigated fitting methods to an offline-trained state-of-the-art logistic classifier with an Ising prior added, which we denote from now on with LOG. Table 4.1 summarizes the mean errors. The results show that the nonlinear fit in average significantly outperforms any linear fitting method as well as the logistic classifier. In our experiments we demonstrated the benefit of learning in conjunction with a regularizer compared to LOG. We show some results on part of the segmented images from the test data chosen by random in Fig. 4.6.

Applying LA

LA is less complex from implementation point of view compared to invLPA as perturbation and fitting are performed in a single step. However, in contrast to invLPA where it is sufficient to solve a linear system, for LA we have to apply the deflected subgradient algorithm, see Algorithm 3.

For the reparametrized potentials (4.35) we chose $\varepsilon = 0.1$ which proofed to be a good choice when using a fixed Ising prior of 1. Note that we choose different Ising priors for the two approaches, depending on which leads to better results. In Fig. 4.8 we illustrate the error we obtain when learning with LA. We compare to the result of LOG. LA outperforms this method in average, see Fig. 4.8 and Table 4.1.

The Ising prior in the logistic classifier prediction step was chosen the same as the approach with which we compared to. The reason for choosing different values for the Ising prior was to illustrate the general benefit of learning with a regularizer. In addition our two approaches invLPA and LA use different solvers for the labeling subproblems. We use MOSEK [ApS15] for invLPA and QPBO [KR07] for LA. Due to this and the different Ising prior for the logistic classifer learning method the results in the plots in Fig. 4.7 and Fig. 4.8 are not identical for the LOG curve.

4.5.3. Learning Pairwise Potentials

In the proceeding section we evaluate the two approaches for learning pairwise potentials when fixing the unary term. For the unary term we simply take the gray scale value minus a thresholding constant, 0.5 in our case. We want to demonstrate that our learning methods are able to learn appropriate pairwise potentials which can preserve to some extent a difficult to segment structure, for which a standard regularizer, e.g. Ising fails completely.

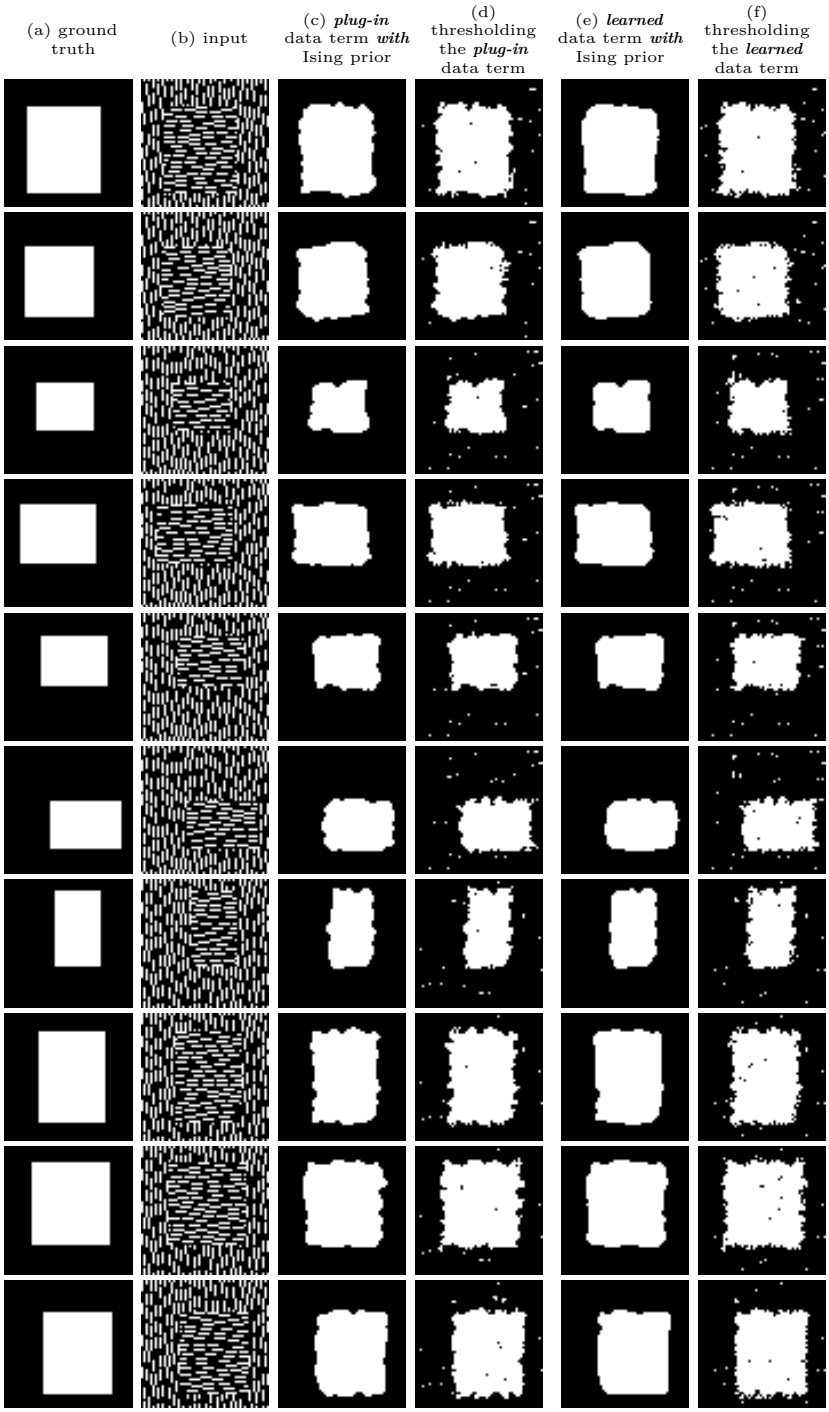


Figure 4.6. **-** Exemplary images from the test dataset chosen at random. (c) Segmentation results when using a linear classifier trained offline using logistic regression with regularizer, denoted by LOG and (d) without additional regularization, in comparison to (e) method invLPA with and (f) without regularizer. The final results improve ***only*** for invLPA that learns ***in conjunction*** with the regularizer.

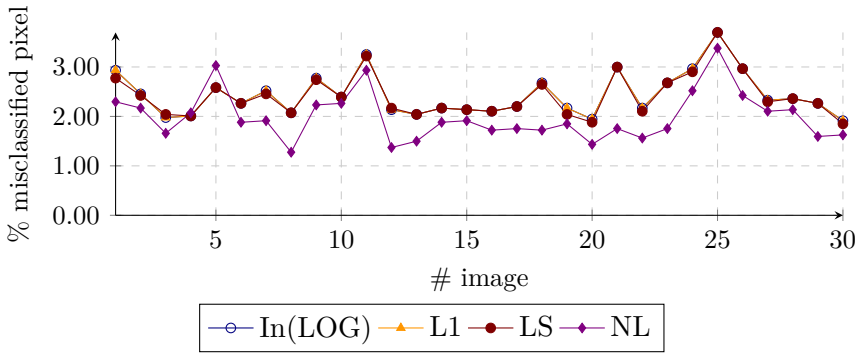


Figure 4.7. **-** Performance on test data of regularization-driven data term learning for the method invLPA with two linear fitting models, linear least-squares fitting (LS), ℓ_1-norm fitting (L1), and nonlinear Gaussian fitting (NL) versus LOG with which we initialize, (ln(LOG)). Remark: the blue and the yellow curve, ln(LOG) and L1 respectively, overlap as the initial $\hat{\theta}$ is already sparse. We can see that the nonlinear Gaussian fitting, NL outperforms the other linear fitting methods. For mean test errors we refer to Table 4.1.

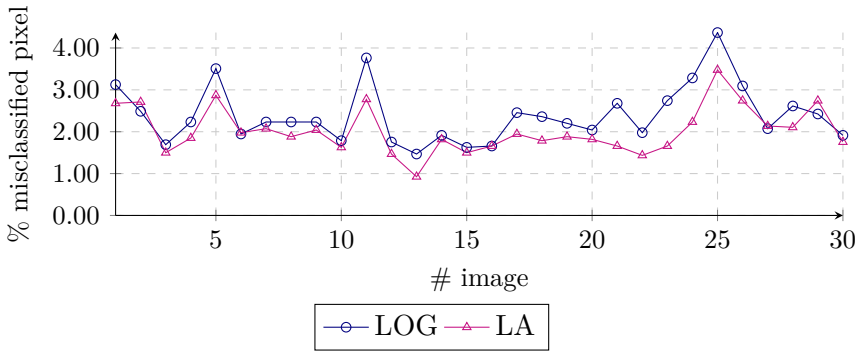


Figure 4.8. **-** Performance on test data of regularization-driven data term learning for the LA method. The plot shows that learning a data term in conjunction with a regularizer with the LA method outperforms LOG. Regarding the mean test errors we refer to Table 4.1.

Experimental Set-Up

In order to validate the two approaches for learning pairwise potentials we use 20 noisy images, evenly split into training and testing sets, with random horizontal and vertical lines, see Fig. 4.9. All the images are of size 64×64 pixels. In order to define our pairwise features we correlate the images with 10 filter masks of different size, see Fig. 4.10, in order to capture the structure of the non equally sized random lines. We define the pairwise features f^{ij}, $ij \in \mathcal{E}$ as matrices

$$f^{ij} := \{G_{\sigma_f}(f^l, f^k)\}_{l,k \in \mathcal{N}(i) \cup \mathcal{N}(j)} \tag{4.103}$$

where each entry is a Gaussian kernel given by $G_{\sigma_f}(f^l, f^k) = \exp(-\frac{1}{2\sigma_f^2}||f^l - f^k||^2)$, with $\sigma_f = 0.4$, and $\mathcal{N}(i)$ and $\mathcal{N}(j)$ define the neighborhoods for the nodes i and j, respectively. We use a 4 neighborhood for the experiments in this subsection. In order to avoid errors due to boundaries handling, we only consider the 56×56 positions where the 5×5 filter masks in Fig. 4.10 completely fit the input data.

Applying invLPA

We describe below the two independent steps, perturbation and fitting.

Perturbation We first obtain initial potentials $\hat{\theta}$, using the LA method. The necessary conversion from the overcomplete to the minimal representation is documented in Appendix A.1.1.

We denote the initial potential from LA with $\hat{\theta}$ and the potentials from invLPA with $\overline{\theta}$. Concerning the notation from Appendix A.1.1 we set the labels $l_1 = 1$ and $l_2 = 0$. We learn only pairwise terms and fix the unary terms $\hat{\theta}^i(1) = p(i) - 0.5$, $\hat{\theta}^i(0) = 0$, where $p(i)$ denotes the gray value of the input image at location i.

Our initial $\hat{\theta}_p$ vector depends linearly on the features and some vector $\hat{w}_p$ we learn using LA and which is given by

$$\hat{w}_p = \begin{pmatrix} \hat{w}^{00} & \hat{w}^{01} \\ \hat{w}^{10} & \hat{w}^{11} \end{pmatrix} \tag{4.104}$$

where we fixed $\hat{w}^{01} = \hat{w}^{10}$ and $\hat{w}^{00} = -20 \cdot \mathbb{1}$ which showed to be a good trade-off between fast-convergence and number of submodular edges. First we convert $\hat{\theta}$ according to (A.12) without changing the optimum value. In the following we use the same notation $\hat{\theta}$ for the new converted model parameters. Now in view of (A.14) we have

$$\overline{\theta}^i = \hat{\theta}^i(1) + \sum_{ij \in \mathcal{E}} \hat{\theta}^{ij}(1,0) + \sum_{ij \in \mathcal{E}} \hat{\theta}^{ij}(0,1) \tag{4.105a}$$

$$\overline{\theta}^{ij} = \hat{\theta}^{ij}(1,1) - \hat{\theta}^{ij}(1,0) - \hat{\theta}^{ij}(0,1). \tag{4.105b}$$

In the first step we compute perturbations θ^i and θ^{ij} for $\overline{\theta}^i$ and $\overline{\theta}^{ij}$ for all nodes and edges from the training set, by solving the linear systems in (4.18) and obtain

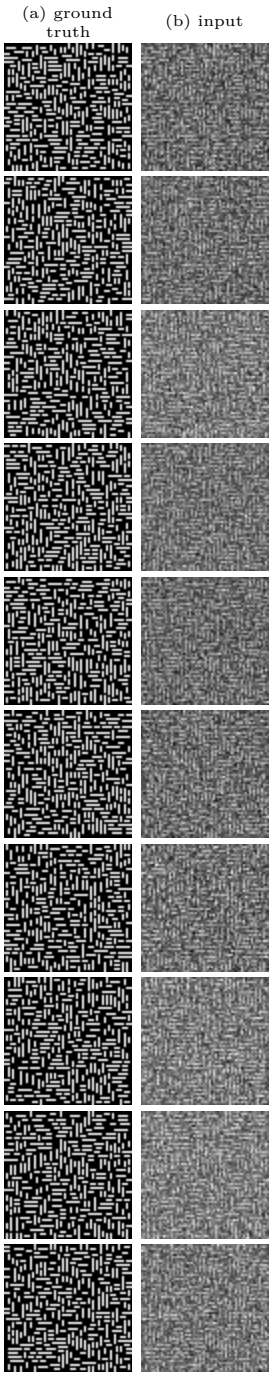


Figure 4.9. **-** We started by creating (a) 10 training images from a random line process as well as (b) noisy versions which were used for learning a regularizer in conjunction with very weak unary potentials (single pixel comparison to a fixed threshold, 0.5 in our case).

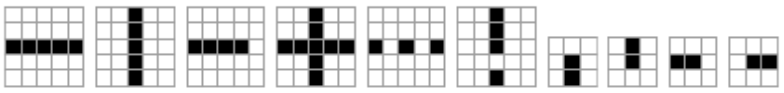


Figure 4.10. - We used 10 binary filter masks for computing feature vectors. After correlating an input image with each of the filters masks, we use neighboring feature vectors for computing Gaussian kernels, in order defining the pairwise features as in (4.103).

the updated parameters

$$\tilde{\theta}^i = \overline{\theta}_i + \theta_i - \hat{\theta}^i \tag{4.106a}$$

$$\tilde{\theta}^{ij} = \overline{\theta}^{ij} + \theta^{ij}, \tag{4.106b}$$

where in the first equation we subtract the initial unary term which we had fixed, $\hat{\theta}^i$, as discussed above.

Fitting The next step consists of fitting w on the perturbed potentials $\tilde{\theta}$. In the following we consider three alternatives:

1. **Linear fitting:** We consider a linear dependency and define $w_u := w^{01} + w^{10}$ and $w_p = w^{11} - w^{01} - w^{10}$.

   - Least square fitting: we solve the following two minimization problems

   $$\min_{w_u} \sum_{i\in[N]} |\langle \sum_{j\in\mathcal{N}(i)} f^{ij}, w_u\rangle - \tilde{\theta}^i| \quad \text{and} \tag{4.107a}$$

   $$\min_{w_p} \sum_{ij\in[M]} |\langle f^{ij}, w_p\rangle - \tilde{\theta}^{ij}| \tag{4.107b}$$

   where f^{ij} are the pairwise features we define as in (4.103), N is the number of all vertices and M the number of all edges from our training data, $\mathcal{N}(i)$ is the defined neighborhood of i, the 4-neighborhood in this case.

   - ℓ_1-norm fitting: we solve

   $$\min_{w_u, s_i} ||w_u||_1 + \lambda \sum_{i\in[N]} |s_i| \quad \text{s.t.} \quad \langle \sum_{j\in\mathcal{N}(i)} f^{ij}, w_u\rangle - s_i = \tilde{\theta}^i \quad \text{and} \tag{4.108a}$$

   $$\min_{w_p, s_{ij}} ||w_p||_1 + \lambda \sum_{ij\in[M]} |s_{ij}| \quad \text{s.t.} \quad \langle f^{ij}, w_p\rangle - s_{ij} = \tilde{\theta}^{ij} \tag{4.108b}$$

   where $\lambda > 0$.

   Please note that the feature vectors in the equations (4.107) and (4.108) correspond to the train data. After learning w_u and w_p we fit them to new feature vectors corresponding to the test data. This defines the new predicted model parameters.

2. **Nonlinear fitting:** Similar to learning unaries, we use the nonlinear predictor function

$$\overline{\theta}(f) := k_M(f)^\top (K(F) + \sigma_m^2)^{-1} \tilde{\theta} \tag{4.109}$$

with

$$k_M(f) = (k(f^1, f), ..., k(f^M, f)) \tag{4.110}$$

where with $f^1, ..., f^M$ are denoted the pairwise features f^{ij} defined in (4.103), M is the number of edges from the training data, σ_m is a parameter obtained with the code from [RW06]. $K(F)$ is defined as $K(F) := (k_{\mathcal{P}}(f^{ij}, f^{kl}))_{ij,kl \in [M]}$ where

$$\begin{aligned} k_{\mathcal{P}}(f^{ij}, f^{kl}) &:= \sigma_{\mathcal{P}_1}^2 \exp\big(-\frac{1}{\sigma_{\mathcal{P}_2}^2} d_p^2(f^{ij}, f^{kl})\big) \quad \text{with} \\ d_{\mathcal{P}}(f^{ij}, f^{kl}) &:= ||\log(f^{ij}) - \log(f^{kl})||_F, \end{aligned} \tag{4.111}$$

where $\sigma_{\mathcal{P}_1}$ and $\sigma_{\mathcal{P}_2}$ are computed using the code in [RW06]. Since we have positive definite matrices for pairwise features, see (4.103) we define the distance $d_{\mathcal{P}}$ as the log-Euclidean metric on the manifold of positive definite matrices.

In Fig. 4.11 we show the error as defined in (4.98) on the test data given in Fig. 4.12(a). We compare the above-mentioned three fitting methods. The results of the outperforming method, least-squares fitting, are illustrated in detail in Fig. 4.12. We used sparse Gaussian regression on 0.1% of all the train data. In Table 4.2 we can observe that invLPA in average improves the initial segmentation which was computed using LA. For all methods the labeling problems are solved using the MOSEK linear solver [ApS15]. As a result we do not need to convert again from minimal to overcomplete representation.

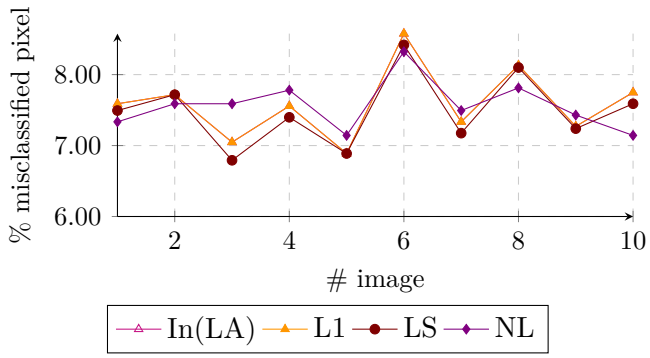


Figure 4.11. **-** Performance on test data of learned pairwise potentials with fixed very weak unary term. Initial potentials are computed with the LA method, (In(LA)), linear fitting with least-squares error (LS), linear fitting with l_1-norm minimization (L1), and Gaussian fitting (NL). We can observe that the invLPA method with least-squares linear fitting in average outperforms the LA method, which was also used to initialize the potentials for invLPA. Remark: The curves for In(LA) and L1 overlap due to sparsity of initial $\hat{\theta}$. For mean errors, see Table 4.2.

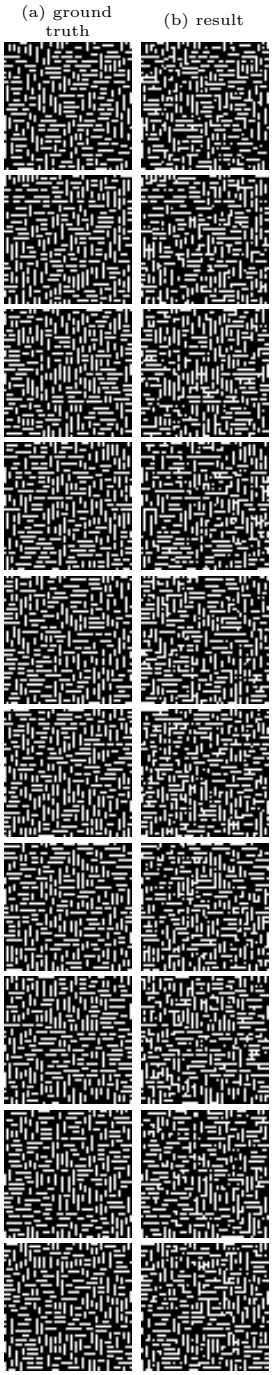


Figure 4.12. - Random line process images. (a) Ground truth segmentations on the test data. (b) Segmentation results when applying the learned regularizer to the noisy input data as in Figure 4.9. The result shows that the regularizer ***captures and preserves*** to some extent the random image structure, unlike any standard prior (e.g. Ising) that is not at all able to discriminate foreground lines from noise.

method	In(LA)	LS	L1	NL
% mis	7.59	**7.48**	7.59	7.56

Table 4.2. - Computed mean test errors from the results in Fig. 4.11. From the results here we conclude that invLPA, ***always*** leads to at least as good or ***improves*** in average on the error of the method that was used to initialize the potentials, In(LA).

Applying LA

We apply the Algorithm 3 while respecting the form of the edge potentials

$$\theta^{ij} = \begin{pmatrix} \langle f^{ij}, w^{00} \rangle & \langle f^{ij}, w^{01} \rangle \\ \langle f^{ij}, w^{10} \rangle & \langle f^{ij}, w^{11} \rangle \end{pmatrix} \tag{4.112}$$

by setting $w^{01} = w^{10}$ and fixing $w^{00} = -20 \cdot \mathbb{1}$. We fix ε for the reparametrized potentials (4.35) to 0.1.

We note that we tried the same scenario with hand-tunning an Ising regularizer. This completely fails by producing constant images, i.e. leads to no segmented random lines.

4.5.4. Experiments on the Weizmann Horse Dataset [BU08]

In this section we illustrate our learning approaches on real world data. We choose a challenging dataset comprising of 327 images of horses with different poses, shades and background including ground truth segmentation. Moreover, we demonstrate the ability of the method LA to efficiently learn unary and pairwise potentials. In addition we compare LA to linear SVM and to two other methods for learning parameters in graphical models.

Experimental Set-Up

We divide the dataset into training and testing dataset. As the images contain horses in different colors we want our training data to be representative enough. We first divide the images into 5 classes, brown, black, white, black-white and brown-white. We randomly divide each of the 5 color type image groups into train (70%) and test (30%) datasets. We used state-of-the-art SLIC [ASS+12] superpixels to process the image data, a common approach to reduce the problem size, and benefit from the rich superpixel structure when extracting features. After resizing the images to the ground truth data size we divide each image into ≈ 800 superpixels.

It is not easy to segment some of the horses images alone without learning due to the same shade of color in the horse area and in the background. Before we group the image pixels into superpixels we preprocess the horse dataset by mapping the image color values in the range from $[\text{red} \quad \text{green} \quad \text{blue}] = [0.2 \quad 0.3 \quad 0]^\top$ to $[0.6 \quad 0.7 \quad 1]^\top$ to the $[0 \quad 0 \quad 0]^\top$ to $[1 \quad 1 \quad 1]^\top$ range.

We have to use more sophisticated features in order to be able to represent and extract the horse shape in all the images. As the horses share the same colors within

Figure 4.13. - A sample horse image from the Weizmann horse dataset [BU08] segmented into SLIC superpixels [ASS+12]. The white lines are the border lines of each superpixel. We colored in red the center superpixel for which we illustrate its three layer neighboring superpixels in green.

the 5 groups, we base our features on color values. Moreover, we tried several color spaces and included those that lead to an improvent of the segmentation result when both training and testing on a separate small subset of the training dataset (3 images). We considered the color spaces HSV, the opponent color space, RGB and CIELAB as features. Only the latter two proved to be informative in our experiments and thus we selected them for defining the unary features. Due to this we used both the RGB and the CIELAB color space when defining the unary features, as well as the pairwise features as we will see in the following.

Both for the unary and pairwise features we use a region covariance descriptor [TPM06]. We can consider our superpixel regions as a Gaussian distribution or a histogram of data. Then for the pairwise features we can use distance measures defined on distributions and histograms.

We use the following superpixel features:

1. Unary features:

$$f^i := \left(LCov^i,\, L^i,\, a^i,\, b^i,\, R^i,\, G^i,\, B^i,\, X^i,\, Y^i\right)^\top. \tag{4.113}$$

The first feature $LCov^i$ is the logarithm of the eigenvalues of the covariance region descriptor [TPM06]. The covariance descriptor of a superpixel S^i is defined using a feature mapping, Φ on the image I such that $\Phi(I, x, y) = F(x, y)$ with

$$F(x,y) := \left(x,\, y,\, L,\, a,\, b,\, R,\, G,\, B,\, |I_x|,\, |I_y|,\, |I_{xx}|,\, |I_{yy}|,\, \sqrt{I_x^2 + I_y^2}\right)^\top. \tag{4.114}$$

We compute the first order (I_x, I_y) and second order image gradients, (I_{xx}, I_{yy}) on the gray scale image I. Then the covariance descriptor on a superpixel S^i

is defined as the covariance matrix of $F(x, y)$ on S^i. The remaining features of f^i are the mean values of pixels on S^i of the CIELAB (L^i, a^i, b^i) and RGB (R^i, G^i, B^i) color space. Furthermore, (X^i, Y^i) are the normalized center coordinates of S^i, motivated by the fact that all horses are approximately located at the image center.

2. Pairwise features:

$$f^{ij} := \left(H^{ij}_{Lab},\, H^{ij}_{RGB},\, CovD^{ij},\, QC^{ij}_{RGB},\, QC^{ij}_{Lab},\, QC^{ij}_{SIFT}\right)^\top, \tag{4.115}$$

where with H we denote the Hellinger distance between multivariate Gaussian distributions. We make the assumption that all pixels p_k in a superpixel S, $p_k \in S$, $k = 1, ..., n$ can be viewed as samples from a Gaussian process with corresponding density functions $P(p|\mu_S, \Sigma_S)$, with μ_S being the 3-dimensional mean vector of all 3-dimensional vector values in some of the color spaces, from all the pixels p_k in the superpixel S and Σ_S being the covariance matrix of the 3-dimensional vectors in the same color space from all the pixels p_k in the superpixel S. Let S^i and S^j be represented by the multivariate normal distributions $\mathcal{N}(\mu_i, \Sigma_i)$ and $\mathcal{N}(\mu_j, \Sigma_j)$ respectively, both estimated from the pixels in the corresponding superpixel. Also let $S^j \in \mathcal{I}_i$, where $\mathcal{I}_i$ is the neighborhood set of superpixel S^i. The Hellinger distance between two multivariate distributions S^i and S^j is defined by

$$H^2(S^i, S^j) := 1 - \frac{\det(\Sigma_i)^{1/4} \det(\Sigma_j)^{1/4}}{\det\left(\frac{\Sigma_i + \Sigma_j}{2}\right)^{1/2}} \exp\left(-\frac{1}{8}(\mu_i - \mu_j)^\top \left(\frac{\Sigma_i + \Sigma_j}{2}\right)^{-1} (\mu_i - \mu_j)\right). \tag{4.116}$$

The color distributions of two superpixel distributions are identical if $H^2 = 0$, and maximally different as $H^2 \to 1$. We define Hellinger distance on the CIELAB and RGB space and denote them with H^{ij}_{Lab} and H^{ij}_{RGB} respectively.

The third pairwise feature $CovD^{ij}$ is the distance on the manifold of the covariance descriptors or positive definite matrices defined by

$$CovD^{ij} := ||\log(Cov^i) - \log(Cov^j)||_F, \tag{4.117}$$

where Cov^i and Cov^j are the region covariance descriptors of the superpixels S^i and S^j respectively, and log stands for the matrix logarithm function. The above defined distance is an approximation of the true geodesic distance which is computationally more expensive.

With QC we denote the quadratic chi squared distance [PW10] between histograms. Both in the RGB and CIELAB space, we compute for every color channel a histogram using 25 bins and normalize the counts to sum up to 1. The quadratic chi histogram distance is normalized using a cross bin similarity

measure which reduces the effect of large bins. For two superpixels S^i and S^j with their histogram like distributions P and Q, respectively, the quadratic chi squared distance is defined as

$$QC_m^A(P,Q) := \sqrt{\sum_{kl}\Big(\frac{P_k - Q_k}{(\sum_c (P_c+Q_c)A_{ck})^m}\Big)\Big(\frac{P_l - Q_l}{(\sum_c (P_c+Q_c)A_{cl})^m}\Big)}, \quad (4.118)$$

and some $0 \leq m < 1$. The cross bin similarity is incorporated through the similarity matrix A defined as

$$A_{kl} := 1 - \frac{D_{kl}}{\max_{kl}(D_{kl})} \quad (4.119)$$

with D_{kl} being cross bin distance between the histogram bin centers. When we have 1D histograms as we consider in our case, D can simply be the ℓ_1 distance across the bins.

We remark that more powerful engineered feature representations may improve the segmentation results. However, as we will show in the evaluation, given inexact potentials our proposed framework learns a more descriptive representation, thus improving the final segmentation result. In order to further improve the numerical efficiency we group superpixels into nonlocal neighborhoods where each group contains the superpixels which are adjacent or connect via up to two superpixels as illustrated in Fig. 4.13.

In addition to error measure (4.98) we also compute the percentage of mislabeled foreground pixels

$$100\frac{Area(F \cap F_{gt})}{Area(F \cup F_{gt})} \quad (4.120)$$

similar to [BYVG11] in order to measure the quality of the foreground mask. Here F and F_{gt} are the computed and ground truth foreground mask, respectively.

Jointly Learning Unary and Pairwise Potentials with LA

For the proposed LA method it is straight forward to apply joint learning of both unary and pairwise potentials, whereas for the invLPA method this is not the case. The reason is the different representation used. For the minimal representation as used in invLPA the unary part of the potentials, see (4.105a) reads for the linear case

$$\overline{\theta}^i = \langle f^i, w^i \rangle + \langle \sum_{ij \in \mathcal{E}} f^{ij}, w^{ij} \rangle. \quad (4.121)$$

After computing the perturbed potentials $\tilde{\theta}^i = \overline{\theta}^i + \theta^i$ it is not possible to clearly relate them to the unary or pairwise features. This is still true for the fitting step, possibly leading to erroneous predictions.

We jointly learn unary and pairwise potentials using the LA method. The unaries are then transfered and fixed for the invLPA method while the pairwise potentials

are used as initialization.

For illustration we demonstrate joint learning with the LA method on the Weizmann horse dataset [BU08] with the setup of features and training images as introduced in Sect. 4.5.4. We define the potentials θ^i and θ^{ij} as in (4.33) with all entries of w different and we fix $\varepsilon = 0.1$. We are aware of the over parametrization when we learn all the entries in our unary and regularizer term. One reason is that this leads to better predicted segmentations. Another reason is to have a fair comparison with the methods from [Dom13] we will compare to in Sect. 4.5.4, which use the above mentioned forms of the potentials. And in order to avoid possible overfitting we add $\frac{||w||^2}{2}$ to the objective in (4.38), where $w = (w^u, w^p)$ as in (4.34).

For comparison we train a linear SVM classifier [SV99] on the unary features. The main reason is to examine how powerful our unary features (4.113) are. We compare joint learning with our LA method with learning the unaries with SVM. This is illustrated in Fig. 4.14 and Fig. 4.15, and report the mean train and test errors in Table 4.3. From the curves in Fig. 4.14 and Fig. 4.15 we can clearly see the significantly better performance of our joint learning approach. This demonstrates that our unary feature vector alone is not sufficiently descriptive to capture the shape and structure of all the horses in the train dataset. However, we can see the benefit of joint learning unary and pairwise terms, in comparison to only learning unaries with linear SVM. Example segmented test images are shown in Fig. 4.16.

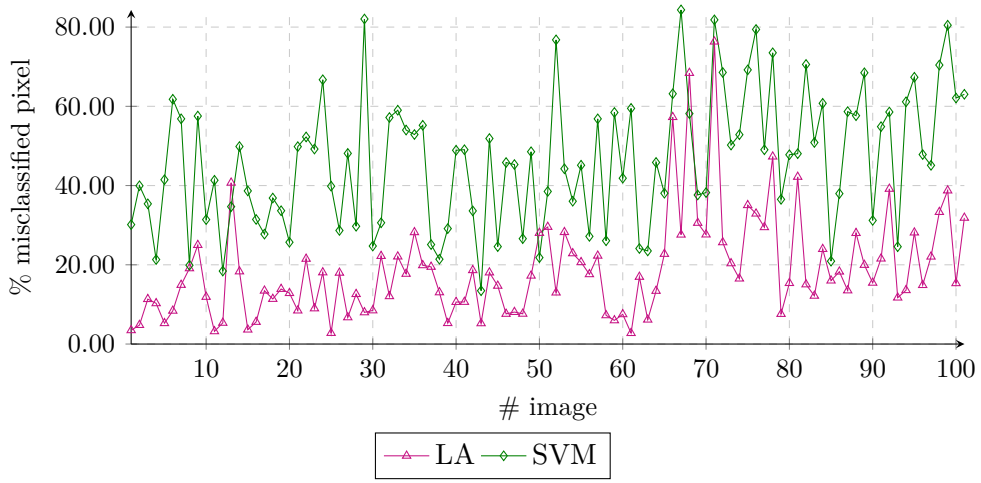


Figure 4.14. - Performance on test data with the error measure (4.98) when learning jointly unary and pairwise terms with LA, compared to training a linear SVM on the unary terms. For mean test errors we refer to Table 4.3.

Learning Pairwise Potentials with invLPA

Next we apply our invLPA method on the Weizmann horse dataset [BU08] to evaluate the capability to learn pairwise terms. As an extension of the comparison of LA and linear SVM we compare our invLPA method to LA with which we initialize. We

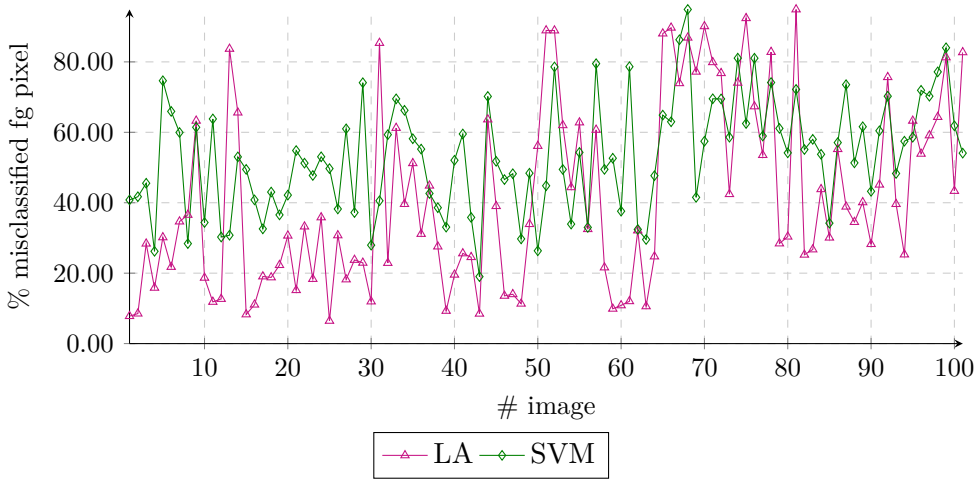


Figure 4.15. - Performance on test data with the error measure (4.120) when learning jointly unary and pairwise terms with LA, compared to training a linear SVM on the unary terms. Concerning mean test errors we refer to Table 4.3.

method	LA	SVM
% mis train	**15.8**	47.0
% mis fg train	**36.8**	53.5
% mis test	**18.7**	46.4
% mis fg test	**41.5**	53.5

Table 4.3. - Comparison of ***jointly learning unary and pairwise*** terms with LA to ***learning unaries only*** with linear SVM. In the table are reported mean train and test errors. The mean test errors correspond to Fig. 4.14 and Fig. 4.15.

transfer the learned unary terms to invLPA and keep them fixed while the pairwise terms are used as initialization.

We apply both linear and nonlinear fitting.

1. **Linear Fitting:** The same as with learning pairwise potentials for the random lines we solve the minimization problems (4.107) for least-squares fitting and (4.108) for ℓ_1-norm fitting.

2. **Nonlinear Fitting:** Same as in 4.5.3 except that now in the Gaussian kernel for measuring the distance of pairwise features we use the ℓ_2-norm as we have vectors of pairwise features. We choose a sparse Gaussian regression with a small subset, 0.3% of all the train data chosen by random, as the sparse Gaussian regression was outperforming the splitting methods described in subsection 4.2.2.

We plot the results in Fig. 4.17 and Fig. 4.18 and provide quantitative train and test errors in Table 4.4. The nonlinear Gaussian fitting outperforms the two linear fitting methods as well as the LA method used to initialize the potentials. However, with

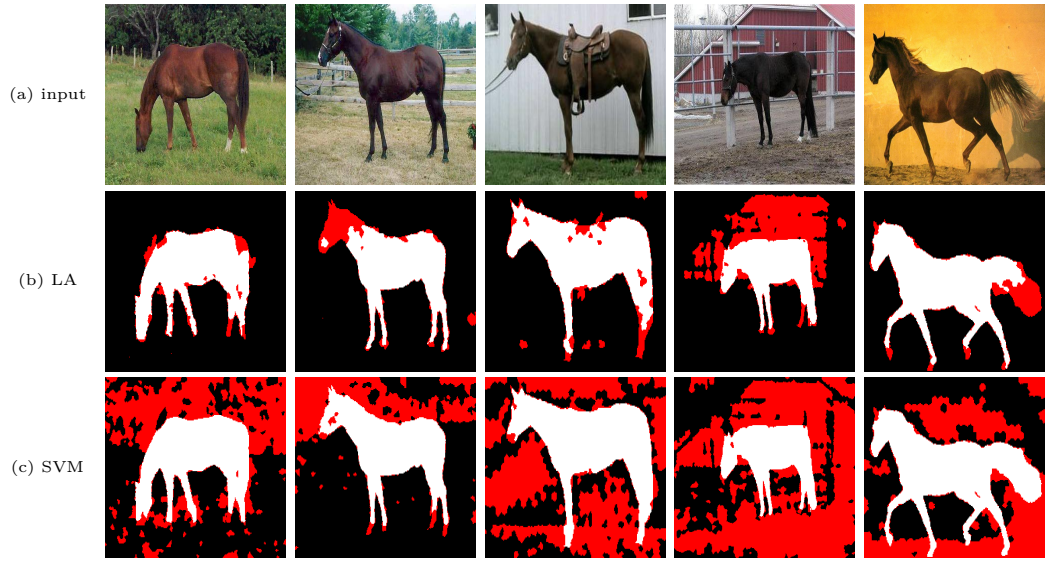


Figure 4.16. - Subset of 5 test images (a) from the Weizmann horse dataset [BU08]. The images (b) are segmented after jointly learning unary and pairwise terms with the LA method and (c) compared to the resulting segmented images when training linear SVM on the unary data.Red color pixels means misclassified pixels. As can be seen from the images joint learning is significantly better than training SVM which has many false positives. These results demonstrate that our unary features alone are not so powerful to capture the structure and pose of all the horses from the train data.

respect to the error measure measuring only mislabeled foreground pixels (4.120) the initial error is not improved, moreover the least-squares fitting as well as the sparse Gaussian regression lead to bigger error than the initial method. The reason for this is the approximate conversion from minimal to overcomplete representation, in order to solve the final labeling problem using the predicted potentials.

Remark 4.5.2. In order to use the results from the LA approach for initialization we first converted from overcomplete to minimal representation. Again for solving the final labeling problem after the fitting procedure we used the same specialized inference solver QPBO [KR07], which we use for LA and requires overcomplete representation of the potentials. Due to the second approximate conversion the results in Table 4.4 for our initialization are slightly different than the ones in Table 4.3. For all results reported here we use the same type of approximate conversion for all fitting methods for invLPA. Concerning the details on the conversions mentioned here we refer to Appendix A.1.1 and Appendix A.1.2.

Comparison of LA to Different Learning Methods

We have already shown the benefit of our learning approaches over state-of-the-art SVM classification method. Now we want to compare it with other learning methods related to our LA method. We note that our invLPA method differs from other standard methods for learning parameters of graphical models. We can think of

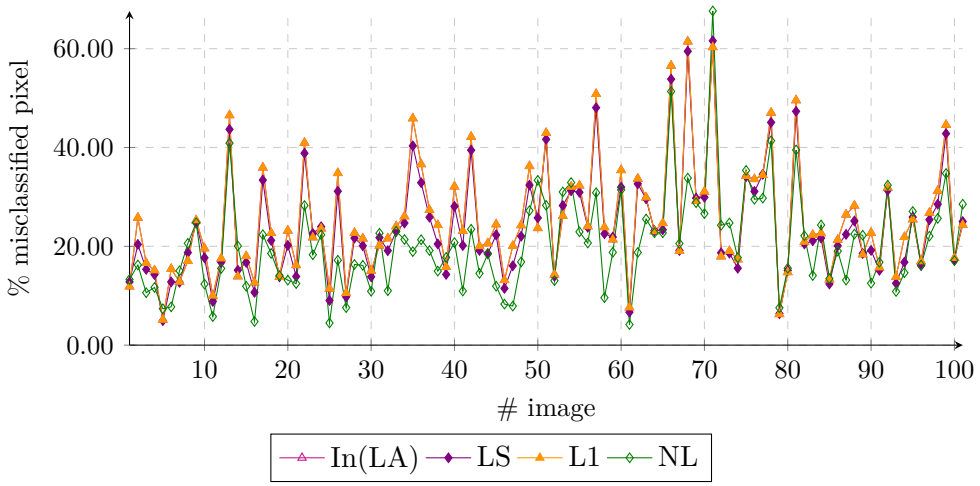


Figure 4.17. - Performance on test data with the error measure (4.98) when learning pairwise terms with invLPA, while initializing with LA (In(LA)). For the fitting step two linear fitting methods are used, least-squares fitting (LS) and ℓ_1-norm linear fitting (L1), as well as a nonlinear Gaussian fitting (NL). From the curves we can see that overall NL performs the best. We remark that In(LA), pink curve and L1, yellow curve overlap due to the sparsity of the initial solution. Concerning mean test errors we refer to Table 4.4.

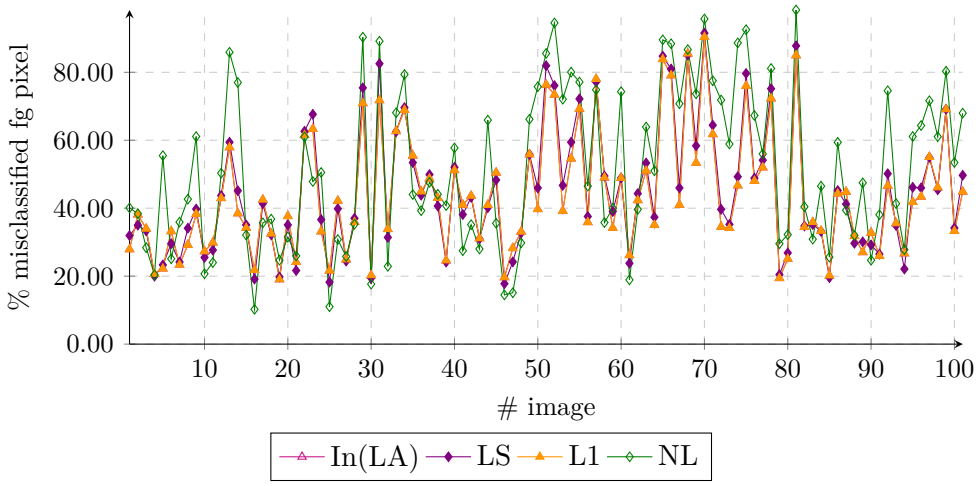


Figure 4.18. - Performance on test data using the error measure as in (4.120) when learning pairwise terms with invLPA, while initializing with LA (In(LA)). For the fitting step two linear fitting methods are used, least-squares fitting (LS) and ℓ_1-norm linear fitting (L1), as well as a nonlinear Gaussian fitting (NL). Again In(LA) and L1, pink and yellow curve respectively overlap. Concerning mean test errors we refer to Table 4.4.

method	In(LA)	LS	L1	NL
% mis train	22.8	21.7	22.8	**18.5**
% mis fg train	**40.8**	41.6	**40.8**	47.6
% mis test	25.1	23.8	25.1	**20.4**
% mis fg test	**43.7**	44.5	**43.7**	51.2

Table 4.4. - Train and test errors when learning pairwise terms with invLPA and initializing with LA (In(LA)) with different fitting methods. Two linear fitting methods, least-squares fitting (LS) and ℓ_1-norm fitting (L1) as well as nonlinear fitting (NL). The mean test errors correspond to the plots in Fig. 4.17 and Fig. 4.18 percentage of mislabeled pixels, % mis test, percentage of mislabeled foreground pixels, % mis fg test.

the invLPA as a method for *correcting* the approximate gradient. In contrast, other methods learn model parameters using approximated gradients.

While learning we minimize a loss function which measures how the energy or distribution defined by the parameter potentials fits the ground truth data. The similarity can be measured in words of Maximum a Posteriori (MAP), Maximum a Posteriori Marginal (MPM) or Maximum Likelihood Estimation (MLE). In the work of Domke [Dom13] learning based on MPM and MLE has been explored and compared against each other. MLE is the most likely estimate, the one which corresponds to the most probable observation. On the contrary MAP estimate is the most probable joint distribution given some observation whereas MPM estimate are the marginally most probable distributions. In other words MPM is MAP for each variable separately.

In [Dom13] Domke in addition to comparing learning methods with different loss functions based on MPM and MLE, proposes a new optimization method, using truncated fitting in order to speed up the inference and a perturbation method for computing the approxiamte gradient. Truncated fitting means optimizing so that an approximate result is reached with a predefined number of iterations. Using a perturbation method for computing the gradient means optimizing twice, the second time initializing with the perturbed output parameters from the first result.

Computing MLE, MAP and MPM estimate in the general case is NP-hard. For this reason there are methods that compute approximate solutions. An older study experimentally shows that the quality of learning depends very much on how the inference method used while learning and the gradient approximation method are coupled [KAH05]. This is the same observation as the one proved in [Wai06].

Learning with MLE was reported more thoroughly in [NL11], while learning with MPM was previously proposed in [Dom12, GRDB06].

One drawback of the comparative study of Domke is that he does not use and compare MAP inference as well as MAP based loss function as in our case. The experimental evaluation in [Dom13] shows that learning benefits from loss functions based on MPM while using MPM inference. The reason can be that while learning we are taking into account the error we get while performing the inference and we measure the quality of each variable (marginal) separately rather than jointly like with MAP or MLE. And as observed in [Dom13] MPM learning happens to be more

robust to model misspecification.

We use the publicly available code from Domke to compare a loss function based on MPM and a loss function based on MLE along with the new proposed optimization procedure in [Dom13], to our LA learning method. We choose the logistic clique loss from MPM like loss functions and surrogate expectation maximization likelihood loss from MLE type of loss functions accompanied with tree reweighed (TRW) message passing algorithm for inference. We choose particularly these two loss functions since in [Dom13] the reported results on the horse dataset suggest that the logistic clique marginal loss performs the best from MPM like loss functions evaluated in [Dom13] and the surrogate likelihood perform the best from the MLE like loss functions evaluated in [Dom13]. For the MLE like loss, when using truncated fitting in the optimization procedure for the inference the prediction results are poor. This is not surprising since the truncated fitting for this type of loss is performed heuristically which is in general true for all truncated fitting methods including the MPM loss. For more details we refer to [Dom13]. Due to this we didn't use truncated fitting for the MLE loss function comparison. Instead, for the reported results for the MLE loss function we used TRW with a convergence threshold of 10^{-5} and regularizer 10^{-3} and 1000 iterations. For the MPM loss function we used truncated fitting with TRW with 5 iterations, regularizer 10^{-3} and 1000 iterations. However, when using TRW with more iterations, e.g. 40 as suggested in [Dom13] for better results we got approximately the same errors as with our LA method, as reported in Table 4.5. The truncated fitting introduced by Domke for our particular setup leads to better results when the inference is less accurate in each step. This leads to doubts for the accuracy of the heuristic optimization method Domke uses, as it appears that increasing the number of iterations deteriorates results instead of improving them.

We used the same experimental set up as for our LA method as described in Sect. 4.5.4. The results are reported in Fig. 4.19 and Fig. 4.20. The mean train and test errors from the comparison are reported in Table 4.5. We can see from the results that the MPM based loss learning method with the logistic clique loss outperforms our method. On the other hand the surrogate likelihood loss from MLE performs much worse than our method. The methods from [Dom13] we compare our learning method with are incomparably slow. On a same machine for the MLE loss the training time was 7 hours and 31 minutes, for MPM for TRW with 5 iterations for what we report our results here, 5 hours and 7 minutes, while the training for our LA method took 1 hour and 48 minutes.

We include some segmented images to illustrate the results in Fig. 4.21.

The results suggest that our LA method is a competitive learning method. Our comparison results also support that when learning parameters in a graphical model the loss function should also correspond to the inference type for better results. MPM loss learning accompanied with inference based on MPM seems to perform the best. However, a reason for the outperformance of the logistic clique loss function can be the heuristic inaccurate optimization used in [Dom13].

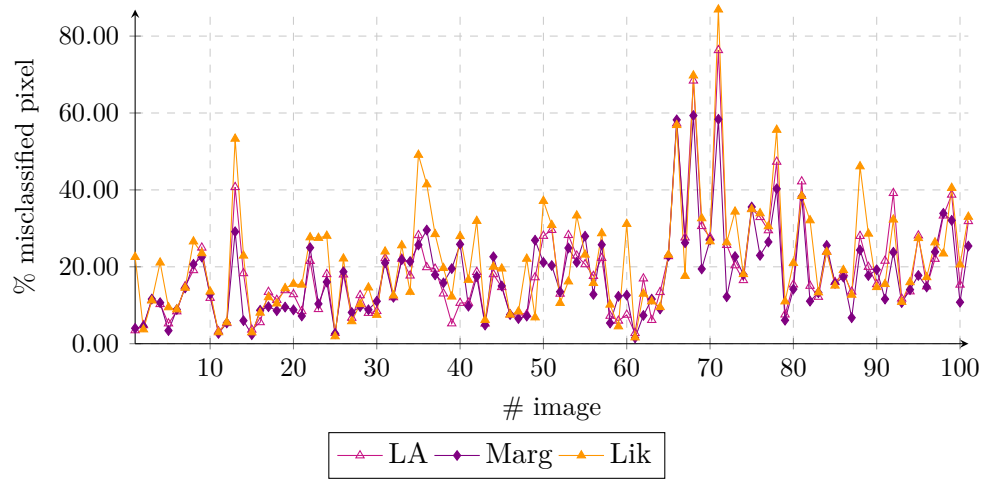


Figure 4.19. **-** Performance on test data with the error measure (4.98) when learning jointly unary and pairwise terms with LA, compared to the learning methods from [Dom13] with a loss function based on MPM, logistic clique loss (Marg) and a loss function based on MLE, surrogate likelihood loss (Lik). The curve for Marg is mostly below the two other curves for LA and Lik, while for some images for instance between 30th and 40th LA is below Marg. Lik on the other hand is almost always having the highest error when compared to LA and Marg. Concerning mean test errors we refer to Table 4.5.

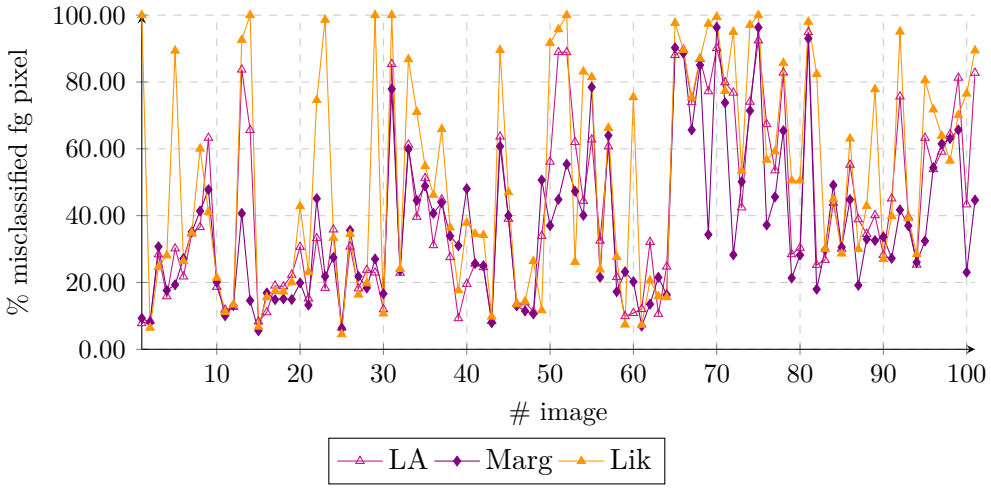


Figure 4.20. **-** Performance on test data with the error measure (4.120) when learning jointly unary and pairwise terms with LA, compared to the learning methods from [Dom13] with a loss function based on MPM, logistic clique loss (Marg) and a loss function based on MLE, surrogate likelihood loss (Lik). Same discussion as in Fig. 4.19. Concerning mean test errors we refer to Table 4.5.

method	LA	Marg	Lik
% mis train	15.8	**14.4**	19.0
% mis fg train	36.8	**31.6**	47.3
% mis test	18.7	**17.4**	22.0
% mis fg test	41.5	**36.5**	52.0

Table 4.5. - Comparison of learning methods on the Weizmann horse dataset. We compare our LA method to two learning methods from Domke [Dom13] one with loss function based on MPM (Marg) and one with loss function based on MLE (Lik). The numbers correspond to the mean train and test errors from Fig. 4.19 percentage of mislabeled pixels, % mis (train) test, and Fig. 4.20, percentage of mislabeled foreground pixels, % mis fg (train) test.

4.5.5. Comparison Between the Two Approaches: invLPA and LA

After performing the experimental evaluation we can empirically compare our two learning methods. In addition we discuss some benefits and drawback of the two proposed learning methods.

We saw that when learning unary potentials in Sect. 4.5.2 invLPA was outperforming LA.

One benefit of LA over invLPA is that it does not require any initialization as we are solving a convex problem and can always obtain a unique global optimum. We enforced uniqueness with the ε term, see (4.30). However, the performance of invLPA depends on the initial potentials we use. This is due to the wrong correction (no correction) for the background segments, as we saw from the experiment in Sect. 4.5.1. From the experimental evaluation we have seen that invLPA always leads to at least as good or better result than the one from the initial model parameters. This was also the case when learning pairwise potentials in Sect. 4.5.3 and initializing with LA, where invLPA lead to improvement in average. Only for learning pairwise potentials on the horse dataset we did not get improvement with respect to the error measure for misclassified foreground nodes. And the reason was the approximate conversion.

The computation time for LA depends on some parameters. In Algorithm 3 we can always set differently the upper bound on the optimal value of the objective $\overline{L}$ as well as n as we pointed out in Remark 4.3.4. Optionally we can add few lines in Algorithm 3 for increasing the upper bound after several iterations. We note that when choosing small parameters like the ones we did for the experiments until now will still lead to convergence as already proven in Theorem 4.3.5 but sometimes it can be very slow.

With the LA method, we can always learn all type of potentials at the same time, unary and pairwise in the case of a pairwise graphical model. However, this is not possible with invLPA due to the minimal representation we use. Due to this we compared only LA with the learning methods from [Dom13].

A very big benefit of invLPA is the freedom to choose any model parameter predictor. We saw that nonlinear predictor like Gaussian which can capture much more from the model structure than simple linear model will almost always result in

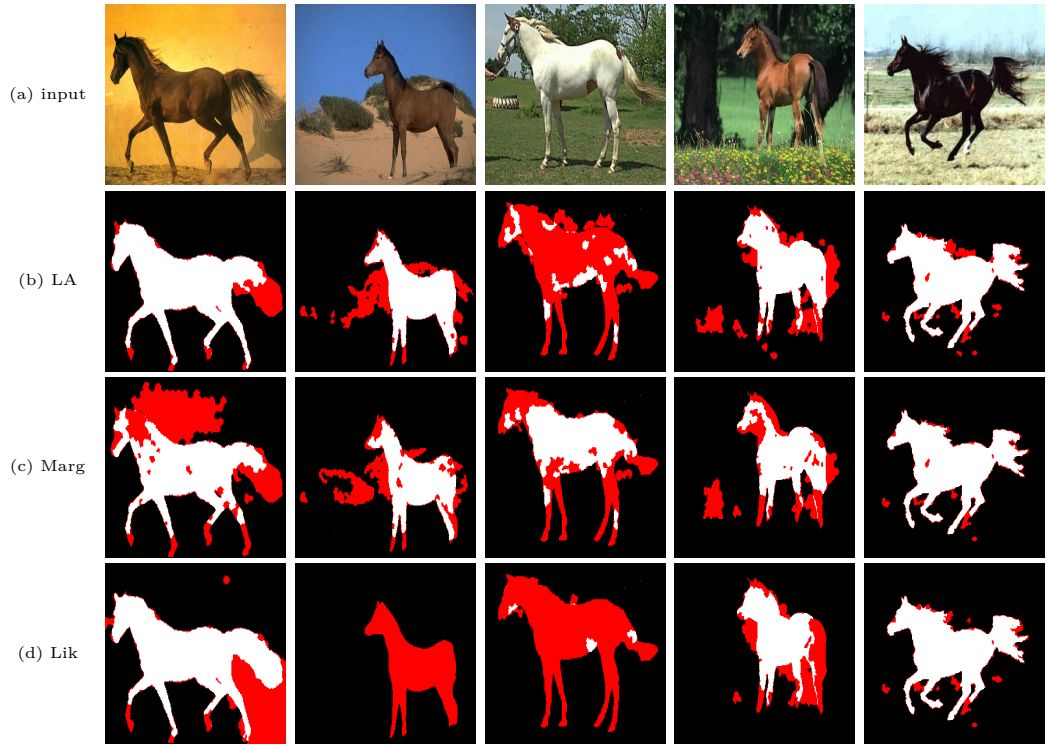


Figure 4.21. - (a) Subset of 5 test images from the Weizmann horse dataset [BU08]. (b) Corresponding resulting segmented images with the LA approach when jointly learning. (c) Segmented images when learning with the method from [Dom13] with a logistic clique marginal loss (Marg) and (d) resulting segmented images when learning with surrogate likelihood loss function from [Dom13] (Lik). Overall Marg performs the best as the mean errors in Table 4.5 suggest. However the results from LA method are very close to those with Marg, as the images here illustrate, although it can happen that for some images LA leads to smaller error, while for others Marg is much better than LA. On the other hand some images are almost not segmented at all with the surrogate likelihood based loss learning method from [Dom13], Lik. Red color denotes misclassified pixels which can be either from the background or foreground.

better predictions than the linear model.

4.6. Semi-Supervised Online Learning in Video Sequences

Our aim is to show another application of our learning methods beside binary segmentation in images. In this section we apply our LA approach for learning motion in dynamic video sequences. By dynamic we mean a video taken by a moving camera in which there are moving objects. To simplify the exposition we consider only one moving object in the sequence. We note that extension to more than one object is possible. We use the ground truth data from only the first three spatio-temporally connected image frames, which is why now our learning is semi-supervised.

We always consider three neighboring frames to be spatio-temporally connected and define spatio-temporal features on these frames. We do ***online learning*** on a video sequence in the following way: starting from the first three frames with provided

ground truth segmentation we learn the motion segmentation for the fourth frame. Afterwards we propagate the learned segmentation of the fourth frame together with the previous two frames forming the spatio-temporal neighborhood and features in order to learn the motion segmentation of the fifth frame and so on.

It is expected that in this way as the number of frames grows the learned segmentation would be inferior. This happens since we learn from "learned" motion segmentations and not from ground truth. Performance can be improved when applying a smoothing stage as post processing in batch mode or online before learning the next frame.

We apply only the LA approach as it is easier from implementation point of view. We note that optionally we can apply the invLPA approach when initializing with LA but in this case we have to perform two steps for each frame, perturbation and fitting, instead of one. In addition we could only learn the unary or the pairwise terms.

Again we use SLIC superpixels [ASS+12] to reduce the problem size and exploiting the richer structure of the superpixels.

Defining the Neighborhood We consider the neighborhood $\mathcal{N}(S_t^i)$ of a superpixel S_t^i in a time frame t as an union of the spatial $\mathcal{N}_{spatial}(S_t^i)$ and spatio-temporal neighborhood, $\mathcal{N}_{spatio-temporal}(S_t^i)$, i.e. $\mathcal{N}(S_t^i) = \mathcal{N}_{spatial}(S_t^i) \cup \mathcal{N}_{spatio-temporal}(S_t^i)$. Below we define the spatial and the spatio-temporal neighborhood.

- Spatial neighborhood, $\mathcal{N}_{spatial}$:

  For spatial neighborhood of a superpixel S^i we consider the closest two layers of superpixels in the same frame t relative to a reference superpixel S_t^i, similarly like for the horse dataset where we considered the three closest layers of a reference center superpixel, see Fig. 4.13. We used less layers due to the larger problem size caused by the image size which is at least five times larger than for the horse dataset.

- Spatio-temporal neighborhood, $\mathcal{N}_{spatio-temporal}$:

  For a superpixel S_t^i in a frame t, its spatio-temporal neighborhood consists of superpixels $S_{t'}^j$ in frame $t' = t \pm 1$ whose spatial position is taken to be the superpixel which contains the pixel corresponding to the superpixel center of S_t^i in the frame t' as well as its spatial neighboring superpixels, as defined above. That is we have

  $$\mathcal{N}_{spatio-temporal}(S_t^i) = S_{t'}^j \cup \mathcal{N}_{spatial}(S_{t'}^j), \tag{4.122}$$

  where the spatial position j in frame t' is obtained such that for the the center pixel of the superpixel S_t^i, $c(S_t^i)$ we have $c(S_t^i) \subset S_{t'}^j$.

Spatio-Temporal Features In addition to using a spatio-temporal neighborhood we want to use spatio-temporal features. We want to have more descriptive representation

of the moving object in the video, which is not always easy to segment taking into account only the features based on appearance.

We define a spatio-temporal region covariance descriptor, similar to the region covariance descriptor we used in Sect. 4.5.4, see (4.114). However, now we have a feature image which is three dimensional, with the third dimension being the time dimension. We define a feature mapping Φ from our video pixel RGB image $I(x, y, t)$ to $F(x, y, t) = \Phi(I, x, y, t)$ with

$$F(x,y,t) := \left(x,\, y,\, t,\, R,\, G,\, B,\, |I_x|,\, |I_y|,\, |I_{xx}|,\, |I_{yy}|,\, \sqrt{I_x^2 + I_y^2},\, \arctan\frac{|I_y|}{|I_x|},\, |I_t|,\, |I_{xt}|,\, |I_{yt}|\right)^\top. \tag{4.123}$$

Compared to the feature mapping for a 2 dimensional image (4.114), we added the derivatives with respect to time t and $\arctan\frac{|I_y|}{|I_x|}$ and excluded the CIELAB color values. In order to compute time derivatives we took the frame before and the frame after for a center image frame of the video and computed the temporal and spatio-temporal derivative images I_t, I_{xt} and I_{yt}.

For the video sequences we used we did not observe a significant improvement when adding the CIELAB color values both for the unary features and for the spatio-temporal region covariance descriptor. Adding $\arctan\frac{|I_y|}{|I_x|}$ in the feature image happened to be a benefit when learning in the video sequences while for the horse images it was not relevant.

After having defined the feature mapping (4.123) for the spatio-temporal region covariance descriptor for unary features we take the logarithm of the eigenvalues of the covariance matrix. For the pairwise spatio-temporal feature we use the same distance measure of covariance matrices on the manifold of positive semi-definite matrices (4.117).

Additional Unary Features The task of semi-supervised learning in a video sequence is more difficult when compared to supervised learning of segmentations from trainining data. Our feature vectors we used for the horse dataset in Sect.4.5.4 are only low-dimensional. We want to extend the unary feature vector for better performance and being able to learn in a semi-supervised manner.

Motivated by the idea of features used in [Dom13] on the horse dataset we use the sinusoidal expansion as in [KOT11]. In particular let us take the subset of our unary features

$$A := \left(X,\, Y,\, R,\, G,\, B\right)^\top \tag{4.124}$$

where X and Y are the normalized x and y center coordinates of the superpixel in the image. We include the features $\sin(bA)$ and $\cos(bA)$, where b is the matrix of all possible binary vectors of length 5. This leads to additional 64 unary features which showed to be beneficial for the segmentation result in practice. Hence, we use them in the following despite their high number of dimensions.

At the end we have a 87 dimensional unary feature vector and 6 dimensional

pairwise feature vector. To sum up we define the unary feature vector of a superpixel S_t^i by

$$f^i := \Big(LCov^i,\, L^i,\, a^i,\, b^i,\, R^i,\, G^i,\, B^i,\, X^i,\, Y^i,\, \sin(bA^i),\, \cos(bA^i)\Big)^\top. \tag{4.125}$$

The unary features are similar to those for the horses (4.113), with difference that now the region covariance descriptor is defined using the feature mapping as in (4.123), and the additional sinusoidal expansion features. A^i is the feature vector as in (4.124) for a superpixel S_t^i in a time frame t. The pairwise features are defined the same as the pairwise features for the horses (4.115) just that now our spatio-temporal region covariance distance is defined using the spatio-temporal region covariance descriptor using the feature mapping as in (4.123).

4.6.1. Experimental Results on the DAVIS Video Dataset [PPTM+16]

The DAVIS video dataset [PPTM+16] comprises of 50 densely segmented video sequences. It is the only dataset to our knowledge that provides dense segmentation of every frame in all the videos. It is a challenging dataset as it includes different type of video sequences, for instance with fast motion, change of appearance, dynamic background, motion blur, occlusions, shape complexity etc. We note that we are interested only in those sequences from DAVIS which include camera movement.

When applying LA we optimized the objective as in (4.38) with the additional term $\frac{||w||^2}{2}$ in order to avoid overfitting which can occur due to the high-dimensionality of the vector w. In order to speed up the learning time we change the parameter in Algorithm 3, the upper bound of the optimal value of the objective $\overline{L} = 10$. We add a few lines in Algorithm 3 in order to increase the upper bound by 5 every 50 iterations of the deflected subgradient algorithm. This can significantly speed up the algorithm. However, for some sequences the execution time for the whole video sequence was still too long. The reason for this is learning from approximate learned segmentation which can be poor after a large number of video frames when the video is more challenging and the object more difficult to segment. As a result on some sequences we did the learning only on part of the sequence.

We evaluate our LA learning method on few sequences taken from the DAVIS dataset [PPTM+16].

As a post processing step after learning the whole or part of the sequence we add smoothing on the pixel level image with contrast sensitive Potts prior [BJ01], where the Potts constant is different for every pair of neighboring pixels and depends on the intensity of the pixels. The contrast sensitive Potts prior is given by

$$P(i,j) = \exp\Big(-\frac{((I(i)-I(j))^2}{2\lambda^2}\Big)\frac{1}{d(i,j)}, \tag{4.126}$$

where i and j are the neighboring pixels with intensities $I(i)$ and $I(j)$ respectively and $d(i,j)$ is the distance between two pixels i and j, which in case of neighboring pixels is 1. For the Potts post processing smoothing we used 4 neighborhood.

In [PPTM+16] the dataset is evaluated using a number of unsupervised, semi-supervised and supervised segmentation algorithms. Some of these algorithms define a graph which can be with higher order potentials and highly connected on the whole video sequence and solve a big optimization problem. The semi-supervised algorithms assume a manual annotation on the first or first few frames, while the supervised algorithms assume that after each frame is segmented, the segmentation is corrected manually and propagated.

There are few error measures used for the evaluation in DAVIS [PPTM+16]. One of the error measures is a Jacardi index which is a total number of mislabeled pixels. This is the same as our error measure (4.98) which gives the percentage of total mislabeled pixels. We compare our results with those with the Jacardi index in [PPTM+16, Table 4]. This is not a fair comparison especially for those videos on which we do not learn the whole sequence as the results in [PPTM+16] are from the whole sequence.

We evaluated starting from the forth frame and stopped when our LA method became too slow. We present the mean errors before and after post processing in Table 4.6. We converted the Jacardi index results in percentage and took the best from all the methods compared in [PPTM+16] independent of whether it is unsupervised, semi-supervised or supervised. The best result from DAVIS to which we compare our results to is usually the one produced using supervised segmentation. Still on the surf sequence on which we evaluate on the whole sequence our method performs best. On car-roundabout the results in [PPTM+16] are a bit better than ours and we perform best on the remaining two sequences on which we did not manage to evaluate on all frames. This comparison results show the benefit of our learning method when compared to even supervised or manually initiated segmentation.

In Fig. 4.22, 4.23, 4.24, 4.25 and 4.26 we present part of the frames from the video sequences we learn together with the obtained segmentation.

sequence	surf(55)	car-round(75)	kite-surf(34)	car-shadow(31)	bus(28)
% mis	5.3	15.5	1.13	9.0	11.0
% mis fg	27.5	25.91	62.0	51.0	35.2
% mis(+Potts)	**4.5**	15.1	**1.11**	**7.0**	**10.0**
% mis fg(+Potts)	24.3	61.7	26.78	45.8	33.3
DAVIS(best)% mis	5.6	**12.9**	34.6	12.0	11.5

Table 4.6. - We compare the results on 5 sequences from the DAVIS video dataset. On the surf and car-roundabout sequence the learning was done on the whole sequence, i.e. 55 and 75 frames, respectively. On the remaining sequences kite-surf, car-shadow and bus only 34 of 50, 31 of 40 and 28 of 80 frames were computed due to time limitations. We use the error measures as in (4.98), % mis and in (4.120), % mis fg. We compare the result before and after post processing with the contrast sensitive Potts prior (4.126). In addition we compare our results % mis to the best reported results from the DAVIS video dataset on the video sequences while using supervised, semi-supervised or unsupervised segmentation algorithms.

As we can see from the reported results and the figures, learning only from the first

three frames using our setup of features is possible only in certain cases when the sequence is not too challenging and the moving object is not so difficult to segment from the rest of the frame. Additional prior information might be enough to improve the learning. Adding more powerful features could also lead to better learning results. Furthermore our invLPA method could be used in the same manner as LA with using only the first three frames with ground truth segmentation. And considering every learned (predicted) subsequent frame for the perurbation step. We expect that this could improve the predicted segmentations.

Our primary aim for performing the experiments on learning in videos was to illustrate another application of our learning methods. And our results indicate competitiveness with the results from the evaluation of DAVIS [PPTM+16].

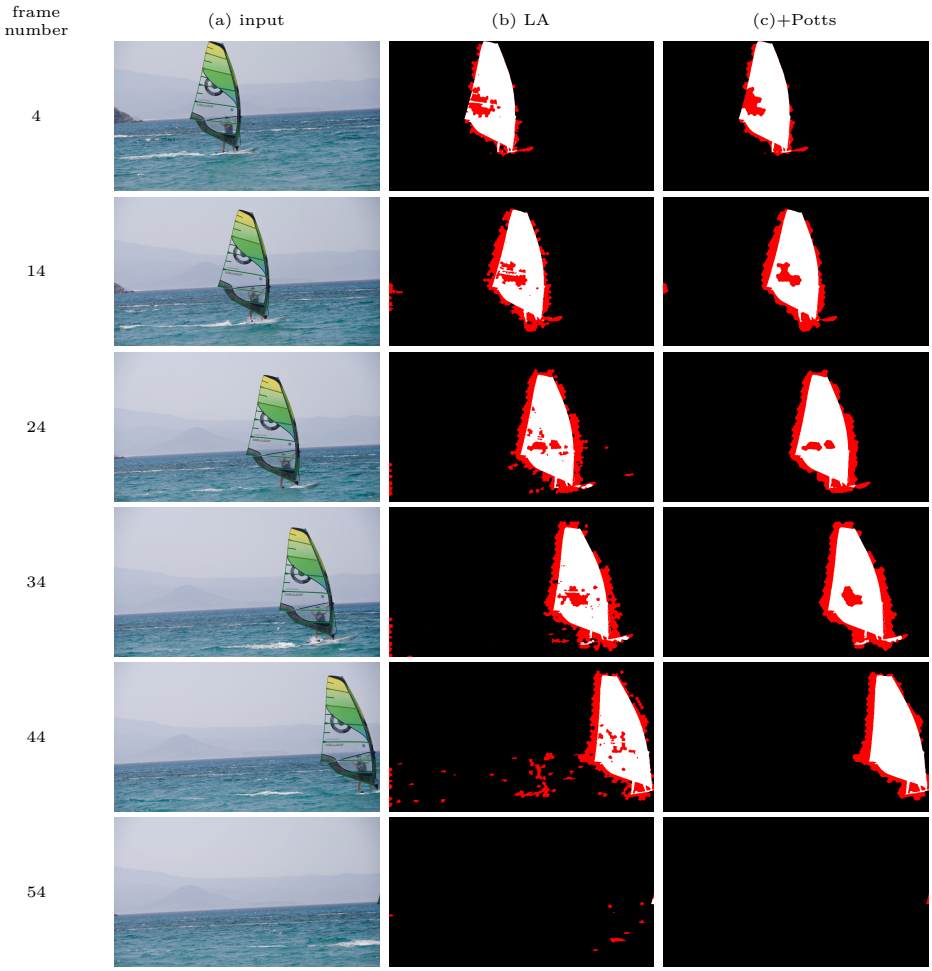


Figure 4.22. - Resulting segmented image frames from the (a) surf sequence with (b) our LA method without post processing and (c) with. Red color denotes misclassified pixels. Our learning method manages to some extent to perform learning through the whole sequence and even the last frame with no surfer is correctly classified. In this sequence there is an movement of the background as the water waves are moving, but still this does not pose a major issue while learning, although in some frames spurious particles of the water are segmented as the surfer.

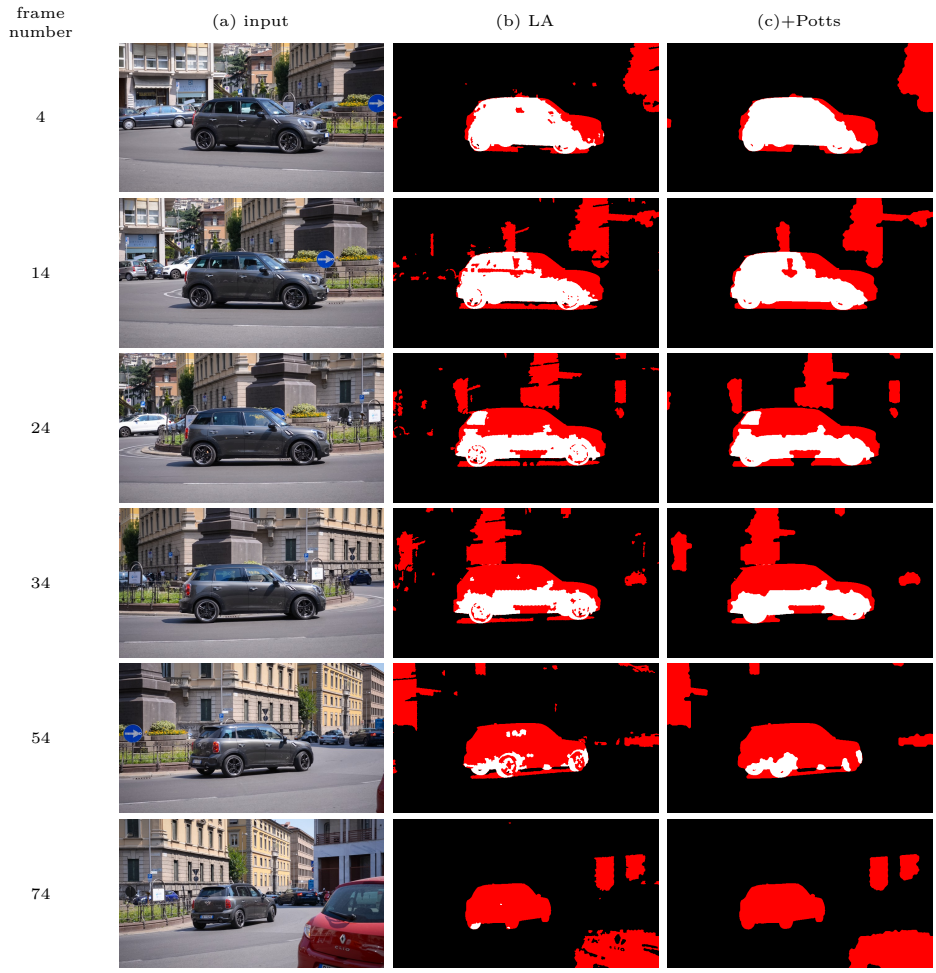


Figure 4.23. - Resulting segmented image frames from (a) the car-roundabout sequence with (a) our LA method without post processing and (c) with. Red color denotes mislabeled pixels. This is a more challenging sequence as we have other objects in the background and in addition there is background clutter. The learning seems to fail at the end when a parked car is suddenly appearing in the sequence. This car has a different color than the rest of the image frame and since most of our features are based on color the algorithm segments the wrong car as a moving one. While our moving car was only partly segmented from the previous frames it is difficult for the algorithm to learn from this inaccurate poor segmentation. We can also observe from the beginning of the sequence that parts of the background are segmented as our moving car, which is due to their similarity in color.

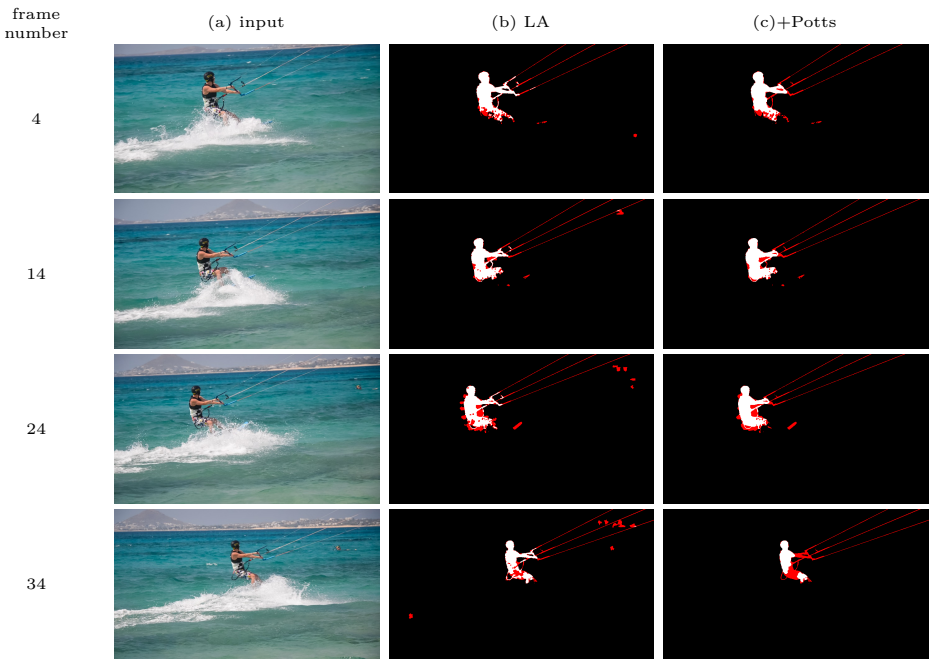


Figure 4.24. - Resulting segmented image frames from (a) the kite-surf sequence with (b) our LA method without post processing with the contrast sensitive Potts prior (4.126) and (c) with post processing. Red color means mislabeled. This is a challenging sequence due to the edge ambiguity, the heterogeneous moving object, occlusion and scale variation. The background water waves are moving, too. Our learning algorithm does not segment the other object, the kite strings to which the surfer is holding as they are too thin. Due to occlusion in the frame 33, the one before the last one illustrated here, part of the surfer in frame 34 is not segmented.

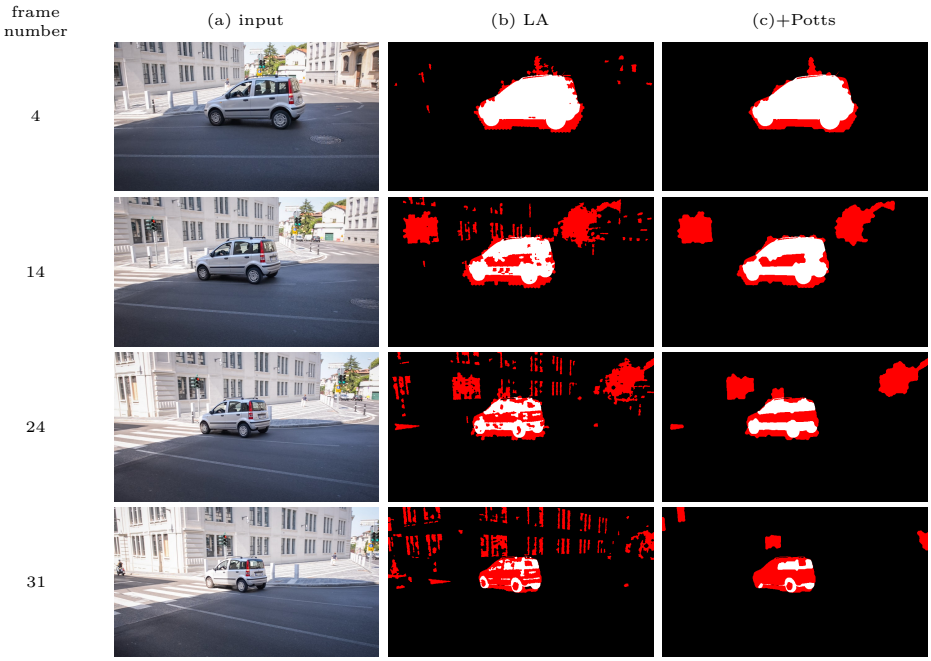


Figure 4.25. **-** Resulting segmented image frames from (a) the car-shadow sequence with (b) our LA method without post processing with the contrast sensitive Potts prior (4.126) and (c) with. Red color means mislabeled. This is a more challenging sequence due to varying illumination. Also the background colors do not differ much from the car colors and after some frames when the segmentation is poorer and the appearance of the car changes, it is more difficult the car to be properly segmented.

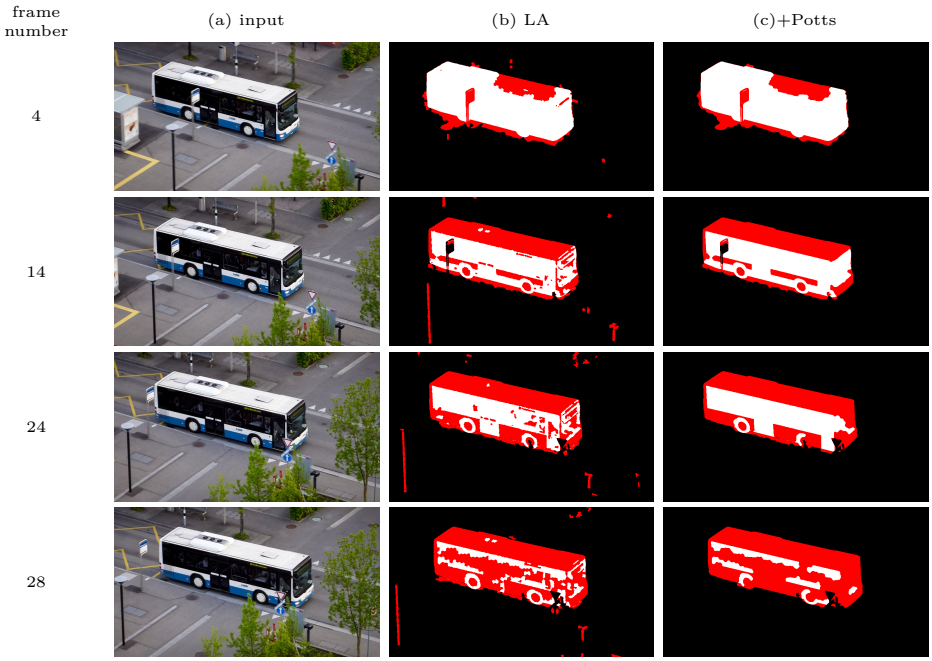


Figure 4.26. - Resulting segmented image frames from (a) the bus sequence with (b) our LA method without post processing with the contrast sensitive Potts prior (4.126) and (c) with. Red color means mislabeled pixels. Again after few frames when the segmentation becomes less accurate it is more difficult to learn from it and only parts of the black color of the bus get segmented.

5. Conclusion and Further Work

In this work we discussed learning on probabilistic graphical models.

Chapter 3 discussed the most common metric learning approaches as well as the optimization methods used for metric learning. Moreover, we proposed suitable optimization technique and applied it to two different metric learning objectives. The experimental results showed competitive performance. Metric learning is interpreted as learning data or unary model parameters based on which a k-means algorithm can perform clustering. Proper distance on the data features is learned which is further used for improving a clustering algorithm.

In our main Chapter 4 we presented our novel learning methods for learning both data (or unary) and pairwise potentials which depend on the features.

Standard learning methods usually learn "approximate" gradients (approximate since relaxed version of the problem is solved) which fit all training data at once. Due to this after learning, the potentials do not lead to ground truth segmentation on all the training data. Motivated by the idea to see how correcting the "approximate" gradients from some learning method can affect on the prediction results we developed a novel learning method very different from present ones in the literature. For this reason we exploited inverse linear programming. With inverse linear programming we can find a perturbation to each of the model parameters so that the final perturbed model parameter will lead to the exact ground truth segmentation on the train data. Our inverse linear programming approach (invLPA) achieves this.

In addition we developed a second novel learning method, the linearized approach (LA) which resembles other existing learning methods, in particular structured SVM. While from the implementation point of view LA is easier to use, it is restricted to linearized potentials.

We demonstrated the advantage of learning when both a unary and a regularizer term is present. Our experiments demonstrated the advantage of learning with our learning methods where we incorporated a fixed regularizer over learning a state-of-the-art logistic classifier offline with a regularizer added in a post processing step. Moreover our experimental evaluation showed that learning a regularizer with a fixed unary term with our novel learning methods can capture and segment a structure for a specific scenario of randomly distributed vertical and horizontal lines on which a standard regularizer e.g. Ising will always fail.

The experiments presented here showed the advantage of learning with inverse linear programming. Moreover, we demonstrated that learning when using exact "approximate" gradients performs at least as good as the method used to initialize the potentials. In fact invLPA always improved on the error over the method used for initialization except in the experiment where we used approximate conversion for solving the final labeling problem with a specialized max-flow solver. In this respect

invLPA outperformed LA as well, when initial potentials for invLPA were learned with LA. Moreover, in the case for learning unary terms where we initialized invLPA with potentials output from training a logistic state-of-the-art classifier, invLPA was better than LA.

We showed the validity of our learning methods on a real world dataset, the Weizmann horse dataset [BU08] and showed their competitiveness when compared to other learning methods for parameters in graphical models. Furthermore, we showed that the application of our learning methods is not restricted to only learning segmentations in images by implementing our LA method to semi-supervised online learning in video sequences, where we considered only the first three frames with ground truth segmentation. This showed promising results in comparison to state-of-the-art segmentation algorithms applied on the video sequences.

One drawback of the invLPA approach is that it only corrects the potentials corresponding to foreground nodes, whereas potentials corresponding to background nodes are not adapted. However, still the corrected potentials lead to the ground truth segmentation. This is a result of the inverse linear program formulation and the representation we use. As a result of only correcting foreground nodes in the first perturbation step of invLPA, the prediction step of the method depends on the initialization of the model parameters.

Further Work

In this work we concentrated on binary image labeling, foreground and background. Next steps could be to extend the invLPA and LA learning methods to the non-binary case. We used only pairwise graphical models and extension to higher order models can be considered as future work as well.

It remains an open question for further work how to extend invLPA to also find corrections for the background segments as well as for pairwise terms in this sense. We believe that using a different representation of the local polytope constraints this drawback of our invLPA method can be fixed. We expect that the results can significantly improve by this step. A further direction of research is to adapt the proposed invLPA method to find corrections that best fit all training data instead of finding corrections on each train data separately.

For our LA method we used a sophisticated subgradient optimization procedure, still much simpler than other optimization methods for non differentiable optimization. Even though the deflected subgradient significantly improves on the speed over using a usual subgradient, more can be done to improve this. In particular, the subgradient method used for our LA method can be replaced by a bundle method for faster speed. Also the extension to non-linear potentials would render LA more flexible.

Concerning online learning motion segmentation in video sequences a lot more can be done using our learning methods. Using additional stronger spatio-temporal features could improve the learning. Incorporating additional step after learning for refining the resulted learned segmentation could enhance the prediction step. Furthermore invLPA could also be used for online learning in video sequences in the

same fashion as LA, using only the first three frames with ground truth segmentation. And considering every prediction from the fourth frame on as a given data (ground truth) which is used for the prediction of the frame to follow and so on.

A. Appendix

A.1. Binary Problems

In this section we describe how to convert an overcomplete parameter representation in energy functions to a minimal representation and vice versa. This is for the purpose of the experimental section of Chapter 4 in order to be able to use the same solver for the labeling problems.

Let us consider minimizing the binary discrete energy

$$\min_{x\in\mathcal{X}} E_\theta(x), \quad E_\theta(x) = \sum_{i\in\mathcal{V}} \theta^i(x_i) + \sum_{ij\in\mathcal{E}} \theta^{ij}(x_i, x_j), \tag{A.1}$$

where every variable $x_i \in \mathcal{X}$ can have one of the two labels in $\mathcal{X} = \{l_1, l_2\}$.

Remark A.1.1. In this work we consider undirected graphs, that is $\theta^{ij}(x_i, x_j) = \theta^{ji}(x_j, x_i)$. Due to this and in order to reduce computational capacity for the representation we assume that $ij \in \mathcal{E}$ implies $ji \notin \mathcal{E}$.

Minimal Representation The encoding of a single variable of the data term by an indicator vector $\mu_i = (\mu_i(l_1), \mu_i(l_2))^\top \in \{(1,0)^\top, (0,1)^\top\}$ reads

$$x_i \to \theta^i(x_i) \in \{\theta^i(l_1), \theta^i(l_2)\} \iff \mu_i \to \langle \theta^i, \mu_i \rangle, \quad \theta^i = (\theta^i(l_1), \theta^i(l_2))^\top. \tag{A.2}$$

Since we consider the binary case we have $\mu_i(l_2) = 1 - \mu_i(l_1)$ and so for a node i we have only one variable $\mu_i(l_1)$. This results in

$$\langle \theta^i, \mu_i \rangle = \theta^i(l_1)\mu_i(l_1) + \theta^i(l_2)(1 - \mu_i(l_1)) = (\theta^i(l_1) - \theta^i(l_2))\mu_i(l_1) + \theta^i(l_2). \tag{A.3}$$

For easier notation we redefine

$$\mu_i := \mu_i(l_1). \tag{A.4}$$

For the pairwise variables we have

$$\theta^{ij} = \theta^{ij}(x_i, x_j) = (\theta^{ij}(l_1, l_1), \theta^{ij}(l_1, l_2), \theta^{ij}(l_2, l_1), \theta^{ij}(l_2, l_2))^\top, \tag{A.5}$$

and for the encoding of each pairwise term by a corresponding indicator vector μ_{ij}

$$(x_i, x_j) \to \theta^{ij}(x_i, x_j) \iff \mu_{ij} \to \langle \theta^{ij}, \mu_{ij} \rangle. \tag{A.6}$$

The labeling decision that the label l_1 is assigned to node i and the label l_2 is assigned to node j is given by $(x_i, x_j) = (l_1, l_2)$ and similarly for the three other cases. These decisions can be encoded using the binary variables for a single node and their

product, that is

$$\mu_{ij} = (\mu_i(l_1)\mu_j(l_1), \mu_i(l_1)\mu_j(l_2), \mu_i(l_2)\mu_j(l_1), \mu_i(l_2)\mu_j(l_2)). \tag{A.7}$$

Using (A.4) and $\mu_i(l_2) = 1 - \mu_i(l_1)$ we can rewrite

$$\mu_{ij} = (\mu_i\mu_j, \mu_i(1-\mu_j), (1-\mu_i)\mu_j, (1-\mu_i)(1-\mu_j))^\top. \tag{A.8}$$

From this follows:

$$\begin{aligned}
\langle \theta^{ij}, \mu_{ij} \rangle =& \theta_{ij}(l_1,l_1)\mu_i\mu_j + \theta^{ij}(l_1,l_2)\mu_i(1-\mu_j) && \text{(A.9a)}\\
&+ \theta^{ij}(l_2,l_1)(1-\mu_i)\mu_j + \theta^{ij}(l_2,l_2)(1-\mu_i)(1-\mu_j) && \text{(A.9b)}\\
=& (\theta^{ij}(l_1,l_1) + \theta^{ij}(l_2,l_2) - \theta^{ij}(l_1,l_2) - \theta^{ij}(l_2,l_1))\mu_i\mu_j && \text{(A.9c)}\\
&+ (\theta^{ij}(l_1,l_2) - \theta^{ij}(l_2,l_2))\mu_i + (\theta^{ij}(l_2,l_1) - \theta^{ij}(l_2,l_2))\mu_j && \text{(A.9d)}\\
&+ \theta^{ij}(l_2,l_2). && \text{(A.9e)}
\end{aligned}$$

We want to have a linearized representation and based on the variables from (A.4) we define

$$\mu_{ij} := \mu_i\mu_j. \tag{A.10}$$

Now using (A.3) and (A.9) and the definitions (A.4) and (A.10) for the discrete energy from (A.1) as a function of μ and dropping terms that do not depend on μ we have

$$\begin{aligned}
E_\theta(\mu) =& \sum_{i\in\mathcal{V}} (\theta^i(l_1) - \theta^i(l_2))\mu_i && \text{(A.11a)}\\
&+ \sum_{ij\in\mathcal{E}} \Big((\theta^{ij}(l_1,l_2) - \theta^{ij}(l_2,l_2))\mu_i + (\theta^{ij}(l_2,l_1) - \theta^{ij}(l_2,l_2))\mu_j && \text{(A.11b)}\\
&+ (\theta^{ij}(l_1,l_1) + \theta^{ij}(l_2,l_2) - \theta^{ij}(l_1,l_2) - \theta^{ij}(l_2,l_1))\mu_{ij} \Big). && \text{(A.11c)}
\end{aligned}$$

Note that adding a constant to every unary and pairwise term will not change the optimum. We choose to subtract $\theta^i(l_2)$ and $\theta^{ij}(l_2,l_2)$ from the unary and pairwise terms repectively and obtain the following new model parameters:

$$\begin{aligned}
\tilde{\theta}^i(l_1) &:= \theta^i(l_1) - \theta^i(l_2) && \text{(A.12a)}\\
\tilde{\theta}^i(l_2) &:= \theta^i(l_2) - \theta^i(l_2) = 0 && \text{(A.12b)}\\
\tilde{\theta}^{ij}(l_1,l_1) &:= \theta^{ij}(l_1,l_1) - \theta^{ij}(l_2,l_2) && \text{(A.12c)}\\
\tilde{\theta}^{ij}(l_1,l_2) &:= \theta^{ij}(l_1,l_2) - \theta^{ij}(l_2,l_2) && \text{(A.12d)}\\
\tilde{\theta}^{ij}(l_2,l_1) &:= \theta^{ij}(l_2,l_1) - \theta^{ij}(l_2,l_2) && \text{(A.12e)}\\
\tilde{\theta}^{ij}(l_2,l_2) &:= \theta^{ij}(l_2,l_2) - \theta^{ij}(l_2,l_2) = 0. && \text{(A.12f)}
\end{aligned}$$

Using the new redefined potentials as in (A.12) and taking into account that $ij = ji$, for the discrete energy in (A.11) we have

$$E_\theta(\mu) = \sum_{i\in\mathcal{V}} (\tilde{\theta}^i(l_1)) + \sum_{ij\in\mathcal{E}} \tilde{\theta}^{ij}(l_1,l_2) + \sum_{ij\in\mathcal{E}} \tilde{\theta}^{ij}(l_2,l_1))\mu_i \tag{A.13a}$$

$$+ \sum_{ij\in\mathcal{E}} (\tilde{\theta}^{ij}(l_1,l_1) - \tilde{\theta}^{ij}(l_1,l_2) - \tilde{\theta}^{ij}(l_2,l_1))\mu_{ij}. \tag{A.13b}$$

A.1.1. Conversion: Overcomplete to Minimal

For our experimental evaluation when initializing the invLPA with LA we have to convert from overcomplete to minimal representation.

Let us denote from now on the model parameters for the overcomplete representation with $\hat{\theta}$ and for the minimal with $\overline{\theta}$. In the overcomplete representation we have given for every node $i \in \mathcal{V}$, $\hat{\theta}^i(l_1)$, $\hat{\theta}^i(l_2)$, and for every edge $ij \in \mathcal{E}$, $\hat{\theta}^{ij}(l_1,l_1)$, $\hat{\theta}^{ij}(l_1,l_2)$, $\hat{\theta}^{ij}(l_2,l_1)$ and $\hat{\theta}^{ij}(l_2,l_2)$. In a first step we can convert them by considering $\theta = \hat{\theta}$ which would also not affect the energy from the overcomplete representation, too. In order to get a minimal representation we need to determine the factors in front of μ_i and μ_{ij} in (A.13) for all nodes i and edges ij and using $\hat{\theta}$ instead of $\tilde{\theta}$ for clarity we get:

$$\overline{\theta}^i = \hat{\theta}^i(l_1) + \sum_{ij\in\mathcal{E}} \hat{\theta}^{ij}(l_1,l_2) + \sum_{ij\in\mathcal{E}} \hat{\theta}^{ij}(l_2,l_1) \tag{A.14a}$$

$$\overline{\theta}^{ij} = \hat{\theta}^{ij}(l_1,l_1) - \hat{\theta}^{ij}(l_1,l_2) - \hat{\theta}^{ij}(l_2,l_1). \tag{A.14b}$$

Given $\hat{\theta}(l_1), \hat{\theta}^{ij}(l_1,l_1), \hat{\theta}^{ij}(l_1,l_2), \hat{\theta}^{ij}(l_2,l_1)$ for all $i \in \mathcal{V}$ and all $ij \in \mathcal{E}$, $\overline{\theta}^i$ and $\overline{\theta}^{ij}$ can be easily computed.

A.1.2. Conversion: Minimal to Overcomplete

Specialized max-flow solvers for labeling problems usually use overcomplete representation and they often produce better segmentation results than a simple linear solver which uses minimal representation. Due to this and a fair comparison, often after prediction we want to solve the labeling problem from the invLPA method with the same max-flow solver we use for LA. For this we need to convert from minimal to overcomplete representation.

However, this is not so straight forward as from overcomplete to minimal. Now given $\overline{\theta}^i$ and $\overline{\theta}^{ij}$, see (A.14) we need to derive $\hat{\theta}$. Without loss of generality we can assume that $\hat{\theta}^i(l_2)$ and $\hat{\theta}^{ij}(l_2,l_2) = 0$. Let us first consider the case when we have any two nodes i and j and the edge between them ij in the minimal representation, that is $\overline{\theta}^i$, $\overline{\theta}^j$ and $\overline{\theta}^{ij}$. This leads to determining 2 variables for the vertices and 3 for the edges in the overcomplete representation. We note that we can use inconsistent labellings, e.g. $\mu_i = 1$, $\mu_j = 0$ and $\mu_{ij} = 1$, because with overcomplete representation consistency is enforced through the constraints rather than the parameterization. Then for two nodes and the edge between them we can have the following label

configurations,

$$\mu \in \{(1,0,0),(0,1,0),(0,0,1),(1,0,1),(0,1,1)\}, \tag{A.15}$$

generating independent equations, while $\{(1,1,0),(1,1,1),(0,0,0)\}$ do not. Then we can derive the following system of equations

$$\hat{\theta}^{i}(l_1) + \hat{\theta}^{ij}(l_1,l_2) + \hat{\theta}^{ij}(l_2,l_1) = \overline{\theta}^{i} \tag{A.16a}$$
$$\hat{\theta}^{j}(l_1) + \hat{\theta}^{ij}(l_1,l_2) + \hat{\theta}^{ij}(l_2,l_1) = \overline{\theta}^{j} \tag{A.16b}$$
$$\hat{\theta}^{ij}(l_1,l_1) - \hat{\theta}^{ij}(l_1,l_2) - \hat{\theta}^{ij}(l_2,l_1) = \overline{\theta}^{ij} \tag{A.16c}$$
$$\hat{\theta}^{i}(l_1) + \hat{\theta}^{ij}(l_1,l_1) = \overline{\theta}^{i} + \overline{\theta}^{ij} \tag{A.16d}$$
$$\hat{\theta}^{j}(l_1) + \hat{\theta}^{ij}(l_1,l_1) = \overline{\theta}^{j} + \overline{\theta}^{ij} \tag{A.16e}$$

The equations above form a linear system $Ax = b$ where

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 & -1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}. \tag{A.17}$$

The matrix A is singular and the system of linear equations is ill posed. This leads to zero or infinite number of solutions. So we do not know if we will (exactly) recover the model parameters for the overcomplete representation. Still we can determine a least-squares solution using Moore-Penrose pseudoinverse [Bje51] of the matrix A.

Similarly, the equation system for all edges and nodes is ill-posed. However, solving this problem in a least-squares sense is computationally expensive and only provides an approximate solution. For this reason in our work we will only use an efficient approximation.

For invLPA in our experimental evaluation we either learn unary or pairwise terms, while fixing the other. We convert from minimal to overcomplete representation when learning the pairwise terms. In this case we know the unary terms and so we only approximately estimate the pairwise terms for the overcomplete representation by solving a linear system as in (A.16).

Bibliography

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale, *The multiplicative weights update method: a meta-algorithm and applications.*, Theory of Computing **8** (2012), no. 1, 121–164.

[AK06] G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*, vol. 147, Springer New York, 2006.

[AO01] R.K. Ahuja and J.B. Orlin, *Inverse Optimization*, Oper. Res. **49** (2001), no. 5, 771–783.

[ApS15] MOSEK ApS, *The MOSEK Optimization Toolbox for MATLAB Manual, version 7.1 (revision 28)*, 2015.

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, *SLIC Superpixels Compared to State-of-the-Art Superpixel Methods*, IEEE Trans. Pattern Anal. Mach. Intell. **34** (2012), no. 11, 2274–2282.

[Bay63] T. Bayes, *An Essay Towards Solving a Problem in the Doctrine of Chances*, Philos. Trans. R.Soc. **35** (1763), 370–418.

[BDS+09] K. Briggman, W. Denk, S. Seung, M. N. Helmstaedter, and S. C. Turaga, *Maximin Affinity Learning of Image Segmentation*, Advances in Neural Information Processing Systems, 2009, pp. 1865–1873.

[Ber99] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, Mass., 1999.

[BHS14] A. Bellet, A. Habhard, and M. Sebban, *A Survey on Metric Learning for Feature Vectors and Structured Data*, Tech. report, Department of Computer Science, University of Southern California, 2014.

[BJ01] Y. Boykov and M-P Jolly, *Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in ND Images*, Proc. IEEE Int'l. Conf. Comput. Vision, vol. 1, 2001, pp. 105–112.

[Bje51] A. Bjerhammar, *Application of Calculus of Matrices to Method of Least Squares with Special Reference to Geodetic Calculations*, Kungl. Tekniska Högskolans Handlingar, Lindståhl, 1951.

[BLSB04] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, *Learning Multi-Label Scene Classification*, Pattern Recognit. **37** (2004), no. 9, 1757–1771.

[Bot98] L. Bottou, *Online Learning and Stochastic Approximations*, Online Learning and Neural Networks **17** (1998), no. 9, 142.

[Boy14] S. Boyd, *Subgradient Methods*, Notes for EE364b, Stanford University Spring (2013-2014).

[BT97] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*, vol. 6, Belmont, MA: Athena Scientific, 1997.

[BT03] A. Beck and M. Teboulle, *Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization*, Oper. Res. Letters **31** (2003), no. 3, 167–175.

[BTN01] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, SIAM, 2001.

[BU08] E. Borenstein and S. Ullman, *Combined Top-Down/Bottom-Up Segmentation*, IEEE Trans. Patt. Anal. Mach. Intell. **30** (2008), no. 12, 2109–2125.

[BV04] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, NY, USA, 2004.

[BVZ98] Y. Boykov, O. Veksler, and R. Zabih, *Markov Random Fields with Efficient Approximations*, Proc. IEEE Conf. Computer Vision Pattern Recognition, June 1998, pp. 648–655.

[BVZ01] ———, *Fast Approximate Energy Minimization via Graph Cuts*, IEEE Trans. Pattern Anal. Mach. Intell. **23** (2001), no. 11, 1222–1239.

[BYVG11] L. Bertelli, T. Yu, D. Vu, and B. Gokturk, *Kernelized Structural SVM Learning for Supervised Object Segmentation*, Proc. IEEE Conf. Computer Vision and Pattern Recognition, June 2011, pp. 2153–2160.

[CBL06] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*, Cambridge University Press, 2006.

[CDLS07] R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*, 1st ed., Springer Publishing Company, Incorporated, 2007.

[CFM75] P.M. Camerini, L. Fratta, and F. Maffioli, *On Improving Relaxation Methods by Modified Gradient Techniques*, Math. Program. Study 3 (1975), 26–34.

[Coo90] G. F. Cooper, *The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks (research note)*, Artif. Intell. **42** (1990), no. 2-3, 393–405.

[CQQT04] X. Chen, H. Qi, L. Qi, and K.-L. Teo, *Smooth Convex Approximation to the Maximum Eigenvalue Function*, J. Global Optim. **30** (2004), no. 2, 253–270.

[dF09] G. d'Antonio and A. Frangioni, *Convergence Analysis of Deflected Conditional Approximate Subgradient Methods*, SIAM J. Optim. **20** (2009), no. 1, 357–386.

[Die12] R. Diestel, *Graph Theory, 4th Edition*, Graduate Texts in Mathematics, vol. 173, Springer, 2012.

[DK14] A. D'Aspremont and N. E. Karoui, *A Stochastic Smoothing Algorithm for Semidefinite Programming*, SIAM J. Optim. **24** (2014), no. 3.

[DKJ+07] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, *Information-Theoretic Metric Learning*, Proc. Int'l. Conf. Mach. Learn. (2007).

[DL09] M. M. Deza and M. Laurent, *Geometry of Cuts and Metrics*, 1st ed., Springer Publishing Company, Incorporated, 2009.

[Dom12] Justin Domke, *Learning Convex Inference of Marginals*, CoRR **abs/1206.3247** (2012).

[Dom13] J. Domke, *Learning Graphical Model Parameters with Approximate Marginal Inference*, IEEE Trans. Pattern Anal. Mach. Intell. **35** (2013), no. 10, 2454–2467.

[EA06] M. Elad and M. Aharon, *Image Denoising via Sparse and Redundant Representations over Learned Dictionaries*, IEEE Trans. Image Process. **15** (2006), no. 12, 3736–3745.

[EFS56] P. Elias, A. Feinstein, and C. Shannon, *A Note on the Maximum Flow Through a Network*, IRE Trans. Inf. Theory **2** (1956), no. 4, 117–119.

[Ete81] N. Etemadi, *An Elementary Proof of the Strong Law of Large Numbers*, Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete **55** (1981), no. 1, 119–122.

[FF56] L. R. Ford and D. R. Fulkerson, *Maximal Flow Through a Network*, Can. J. Math. **8** (1956), 399–404.

[Fis36] R.A. Fisher, *The Use of Multiple Measurements in Taxonomic Problems*, Annals of Eugenics **7** (1936), no. 2, 179–188.

[FJ08] T. Finley and T. Joachims, *Training Structural SVMs when Exact Inference is Intractable*, Proc. Int'l. Conf. Machine Learning, 2008.

[FW56] M. Frank and P. Wolfe, *An Algorithm for Quadratic Programming*, Nav. Res. Logist. **3** (1956), no. 1-2, 95–110.

[GB] M. Grant and S. Boyd, *CVX: MATLAB Software for Disciplined Convex Programming*.

[GH11] D. Garber and E. Hazan, *Approximating Semidefinite Programs in Sublinear Time*, Advances in Neural Infromation Processing Systems **24** (2011), 1080–1088.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Algorithms and Combinatorics, vol. 2, Springer, 1988 (English).

[GRDB06] S. S. Gross, O. Russakovsky, C. B. Do, and S. Batzoglou, *Training Conditional Random Fields for Maximum Labelwise Accuracy*, Proc. 19th Int'l. Conf. Neural Information Processing Systems (Cambridge, MA, USA), MIT Press, 2006, pp. 529–536.

[Gut03] B. Guta, *Subgradient Optimization Methods in Integer Programming with an Application to a Radiation Therapy Problem*, Ph.D. thesis, Technische Universität Kaiserslautern, 2003.

[GVL96] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.

[Haz08] E. Hazan, *Sparse Approximate Solutions to Semidefinite Programs*, Proc. 8th Latin Amer. Symp. on Theoretical Informatics, Springer, 2008, pp. 306–316.

[HC71] J. M. Hammersley and P. E. Clifford, *Markov Random Fields on Finite Graphs and Lattices*, *Unpublished manuscript* (1971).

[Hei03] B. Heisele, *Visual Object Recognition with Supervised Learning*, IEEE Intell. Syst. **18** (2003), no. 3, 38–42.

[HHS84] P. L. Hammer, P. Hansen, and B. Simeone, *Roof Duality, Complementation and Persistency in Quadratic 0–1 Optimization*, Math. Program. **28** (1984), no. 2, 121–155.

[HKWL14] D. A. Huang, L. W. Kang, Y. C. F. Wang, and C. W. Lin, *Self-Learning Based Image Decomposition with Applications to Single Image Denoising*, IEEE Trans. Multimedia **16** (2014), no. 1, 83–93.

[HR00] C. Helmberg and F. Rendl, *A Spectral Bundle Method for Semidefinite Programming*, SIAM J. Optim. **10** (2000), no. 3, 673–696.

[HUL93] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms I*, Springer Berlin Heidelberg, 1993.

[Ish03] H. Ishikawa, *Exact Optimization for Markov Random Fields with Convex Priors*, IEEE Trans. Pattern Anal. Mach. Intell. **25** (2003), no. 10, 1333–1336.

[Isi25] E. Ising, *Beitrag zur Theorie des Ferromagnetismus*, Zeitschrift für Physik **31** (1925), no. 1, 253–258.

[JFY09] T. Joachims, T. Finley, and C.-N. J. Yu, *Cutting-Plane Training of Structural SVMs*, Machine Learning **77** (2009), no. 1, 27–59.

[JKDG08] P. Jain, B. Kulis, I. Dhillon, and K. Grauman, *Online Metric Learning and Fast Similarity Search*, Advances in Neural Information Processing Systems (2008).

[KAH05] Sanjiv Kumar, Jonas August, and Martial Hebert, *Exploiting Inference for Approximate Parameter Learning in Discriminative Fields: An Empirical Study*, pp. 153–168, Springer Berlin Heidelberg, 2005.

[Kar84] N. Karmarkar, *A New Polynomial-Time Algorithm for Linear Programming*, Proc. 16th Annual ACM Symposium on Theory of Computing, ACM, 1984, pp. 302–311.

[KF09] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, 2009.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by Simulated Annealing*, Science **220** (1983), no. 4598, 671–680.

[KL51] S. Kullback and R. A. Leibler, *On Information and Sufficiency*, The Anals of Mathematical Statistics **22** (1951), no. 1, 79–86.

[Kol06] V. Kolmogorov, *Convergent Tree-Reweighted Message Passing for Energy Minimization*, IEEE Trans. Pattern Anal. Mach. Intell. **28** (2006), no. 10, 1568–1583.

[KOT11] G. Konidaris, S. Osentoski, and P. S. Thomas, *Value Function Approximation in Reinforcement Learning Using the Fourier Basis*, AAAI Conf. Artificial Intelligence **6** (2011).

[KR07] V. Kolmogorov and C. Rother, *Minimizing Non-Submodular Functions with Graph Cuts - A Review*, IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007), no. 7, 1274–1279.

[KT07] Nikos Komodakis and Georgios Tziritas, *Approximate Labeling via Graph Cuts Based on Linear Programming*, IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007), no. 8, 1436–1453.

[Kul12] B. Kulis, *Metric Learning: A Survey.*, Foundations and Trends in Machine Learning **5** (2012), no. 4, 287–364.

[KZ04] V. Kolmogorov and R. Zabin, *What Energy Functions can be Minimized via Graph Cuts?*, IEEE Trans. Pattern Anal. Mach. Intell. **26** (2004), no. 2, 147–159.

[Lau96] S. L. Lauritzen, *Graphical Models*, Oxford University Press, 1996.

[LG12] O. Lézoray and L. J Grady, *Image Processing and Analysis with Graphs, Theory and Practice*, CRC Press, 2012.

[LH05] Y. LeCun and F. J. Huang, *Loss Functions for Discriminative Training of Energy-Based Models*, AlStats., January 2005.

[LHB04] Y. LeCun, F. J. Huang, and L. Bottou, *Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting*, Proc. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, CVPR, vol. 2, IEEE, 2004, pp. II–104.

[Lic13] M. Lichman, *UCI Machine Learning Repository*, 2013.

[Llo82] S. Lloyd, *Least Squares Quantization in PCM*, IEEE Trans. Inf. Theory **28** (1982), no. 2, 129–137.

[LO96] A. S. Lewis and M. L. Overton, *Eigenvalue Optimization*, Acta Numerica **5** (1996), 149–190.

[Lov83] L. Lovász, *Submodular Functions and Convexity*, Mathematical Programming The State of the Art, Springer, 1983, pp. 235–257.

[LP66] E. S. Levitin and B. T. Polyak, *Constrained Minimization Methods*, USSR Comput. Math. Math. Phys. **6** (1966), no. 5, 1–50.

[LS03] S. L. Lauritzen and N. A. Sheehan, *Graphical Models for Genetic Aanalyses*, Statistical Science (2003), no. 18, 489–514.

[Lux07] Ulrike Luxburg, *A Tutorial on Spectral Clustering*, Statistics and Computing **17** (2007), no. 4, 395–416.

[Mah36] P. C. Mahalanobis, *On the Generalized Distance in Statistics*, Proc. National Institute of Sciences (Calcutta) **2** (1936), 49–55.

[MWJ99] K. P. Murphy, Y. Weiss, and M. I. Jordan, *Loopy Belief Propagation for Approximate Inference: An Empirical Study*, Proc. 15th Conf. Uncertainty in Artificial Intelligence (San Francisco, CA, USA), UAI'99, Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[ND14] J. W. Ng and M. P. Deisenroth, *Hierarchical Mixture-of-Experts Model for Large-Scale Gaussian Process Regression*, ArXiv e-prints (2014).

[Ned08] A. Nedich, *Subgradient Projection Method*, Lecture Notes on Convex Analysis IE 598 AN Fall 2008, 2008.

[Nes04] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Kluwer Academic Publishers, 2004.

[Nes05] ———, *Smooth Minimization of Non-Smooth Functions*, Math. Program. **103** (2005), no. 1, 127–152.

[NG08] N. Nguyen and Y. Guo, *Metric Learning: A Support Vector Approach*, European Conf. Mach. Learn. Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2008), 125–136.

[NJW01] A.Y. Ng, M.I. Jordan, and Y. Weiss, *On Spectral Clustering: Analysis and an Algorithm*, Advances In Neural Information Processing Systems, MIT Press, 2001, pp. 849–856.

[NL11] S. Nowozin and C. H. Lampert, *Structured Learning and Prediction in Computer Vision*, Foundations and Trends in Computer Graphics and Vision **6** (2011), no. 3–4, 185–365.

[NT08] A. S. Nemirovski and M. J. Todd, *Interior-Point Methods for Optimization*, Acta Numerica **17** (2008), 191–234.

[NW06] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer, New York, 2006.

[Ous00] F. Oustry, *A Second-Order Bundle Method to Minimize the Maximum Eigenvalue Function*, Math. Program. **89** (2000), no. 1, 1–33.

[Ove88] M. L. Overton, *On Minimizing the Maximum Eigenvalue of a Symmetric Matrix*, SIAM J. Matrix Anal. Appl. **9** (1988), no. 2, 256–268.

[OW93] M. L. Overton and R. S. Womersley, *Optimality Conditions and Duality Theory for Minimizing Sums of the Largest Eigenvalues of Symmetric Matrices*, Math. Program. **62** (1993), no. 1-3, 321–357.

[PKT11] N. Paragios, N. Komodakis, and Tziritas, *MRF Energy Minimization and Beyond via Dual Decomposition*, IEEE Trans. Pattern Anal. Mach. Intell. **33** (2011), 531–552.

[Pol69] B.T Polyak, *Minimization of Unsmooth Functionals*, U.S.S.R. Comput. Math. Math. Phys. **9** (1969), 14–29.

[Pot52] R. B. Potts, *Spontaneous Magnetization of a Triangular Ising Lattice*, Phys. Rev. **88** (1952), 352–352.

[PPTM+16] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung, *A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation*, Comput. Vision Patt. Recognition, 2016.

[PW00] F. A. Potra and S. J. Wright, *Interior-Point Methods*, J. Comput. Appl. Math. **124** (2000), no. 1, 281–302.

[PW10] O. Pele and M. Werman, *The Quadratic-Chi Histogram Distance Family*, European Conf. Comput. Vision, 2010, pp. 749–762.

[RM03] X. Ren and J. Malik, *Learning a Classification Model for Segmentation*, Proc. 9th IEEE Int'l Conf. Comput. Vision, IEEE, 2003, pp. 10–17.

[Roc70] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, 1970.

[RW06] C.E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.

[Sch98] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, 1998.

[SF95] A. Shapiro and M. K. H. Fan, *On Eigenvalue Optimization*, SIAM J. Optim. **5** (1995), no. 3, 552–569.

[Shi94] S. E. Shimony, *Finding MAPs for Belief Networks is NP-Hard*, Artif. Intell. **68** (1994), no. 2, 399–410.

[Shl76] M. I. Shlesinger, *Syntactic Analysis of Two-Dimensional Visual Signals in the Presence of Noise*, Kibernetika **4** (1976), 113–130.

[Sho85] N. Z. Shor, *Minimization Methods for Non-Differentiable Functions*, vol. 3, Springer Science and Business Media, 1985.

[SK86] T. P. Speed and H. T. Kiiveri, *Gaussian Markov Distributions over Finite Graphs*, The Annals of Statistics **14** (1986), 138–150.

[SM00] J. Shi and J. Malik, *Normalized Cuts and Image Segmentation*, IEEE Trans. Pattern Anal. Mach. Intell. **22(8)** (2000), 888–905.

[SS01] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, USA, 2001.

[SSKS12] B. Savchynskyy, S. Schmidt, J.H. Kappes, and C. Schnörr, *Efficient MRF Energy Minimization via Adaptive Diminishing Smoothing*, Uncertainty in Artificial Intelligence (2012), 746–755.

[SSM98] B. Schölkopf, A. Smola, and K.-R. Müller, *Nonlinear Component Analysis as a Kernel Eigenvalue Problem*, Neural Comput. **10** (1998), no. 5, 1299–1319.

[SSSN04] S. Shalev-Shwartz, Y. Singer, and Y. N. Ng, *Online Learning of Pseudo-Metrics*, Proc. International Conf. Mach. Learn. (2004).

[ST02] A. Schwaighofer and V. Tresp, *Transductive and Inductive Methods for Approximate Gaussian Process Regression*, Advances in Neural Information Processing Systems 15 (S. Thrun and K. Obermayer, eds.), MIT Press, Cambridge, MA, 2002, pp. 953–960.

[SV99] J. A. K. Suykens and J. Vandewalle, *Least Squares Support Vector Machine Classifiers*, Neural Process. Lett. **9** (1999), no. 3, 293–300.

[THJA04] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, *Support Vector Machine Learning for Interdependent and Structured Output Spaces*, Proc. 21st Int'l. Conf. Mach. Learn., ACM, 2004, pp. 104–111.

[TJHA05] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, *Large Margin Methods for Structured and Interdependent Output Variables*, J. Mach. Learn. Res. **6** (2005), 1453–1484.

[Tod01] M. J. Todd, *Semidefinite Optimization*, Acta Numerica 2001 **10** (2001), 515–560.

[TPM06] O. Tuzel, F. Porikli, and P. Meer, *Region Covariance: A Fast Descriptor for Detection and Classification*, European Conf. Comput. Vision, 2006, pp. 589–600.

[Tre00] V. Tresp, *A Bayesian Committee Machine*, Neural Computation **12** (2000), 2000.

[TSÅP17] V. Trajkovska, P. Swoboda, F. Åström, and S. Petra, *Graphical Model Parameter Learning by Inverse Linear Programming*, LNCS 10302, pp. 323–334, Springer, 2017.

[Wai06] M.J. Wainwright, *Estimating the "Wrong" Graphical Model: Benefits in the Computation-Limited Setting*, J. Mach. Learning Res. **7** (2006), 1829–1859.

[WBS06] K. Q. Weinberger, J. Blitzer, and L. Saul, *Distance Metric Learning for Large Margin Nearest Neighbor Classification*, In Advances in Neural Information Processing Systems (NIPS) (2006), no. 18, 1473?1480.

[Wer07] T. Werner, *A Linear Programming Approach to Max-sum Problem: A Review*, IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007), no. 7, 1165–1179.

[WIC12] T. Windheuser, H. Ishikawa, and D. Cremers, *Generalized Roof Duality for Multi-Label Optimization: Optimal Lower Bounds and Persistency*, European Conf. Comput. Vision, Springer, 2012, pp. 400–413.

[WJ08] M. J Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*, Now Publishers Inc., 2008.

[WJW02] Martin Wainwright, Tommi Jaakkola, and Alan Willsky, *MAP Estimation via Agreement on (Hyper)trees: Message-Passing and Linear Programming Approaches*, IEEE Trans. Information Theory **51** (2002), 3697–3717.

[WJW05a] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, *A New Class of Upper Bounds on the Log Partition Function*, IEEE Trans. Inf. Theory **51** (2005), no. 7, 2313–2335.

[WJW05b] ———, *MAP Estimation via Agreement on Trees: Message-Passing and Linear Programming*, IEEE Trans. Inf. Theory **51** (2005), no. 11, 3697–3717.

[WS08] K. Q. Weinberger and L. K. Saul, *Fast Solvers and Efficient Implementations for Distance Metric Learning*, Proc. 25th int'l. Conf. Mach. Learn., ACM, 2008, pp. 1160–1167.

[WS09] ———, *Distance Metric Learning for Large Margin Nearest Neighbor Classification*, J. Mach. Learn. Research **10** (2009), 2017–244.

[XNJR02] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell, *Distance Metric Learning with Application to Clustering with Side-Information*, In Advances in Neural Information Processing Systems **15** (2002), 505–512.

[YFW05] J. S. Yedidia, W. T. Freeman, and Y. Weiss, *Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms*, IEEE Trans. Inf. Theory **51** (2005), no. 7, 2282–2312.

[YL12] Y. Ying and P. Li, *Distance Metric Learning with Eigenvalue Optimization*, J. Mach. Learn. Research **13** (2012), 1–26.

[Zin03] M. Zinkevich, *Online Convex Programming and Generalized Infinitesimal Gradient Ascent*, Proc. International Conf. Mach. Learn. (2003), 928–936.

[ZL96] J. Zhang and Z Liu, *Calculating Some Inverse Linear Programming Problems*, J. Comput. Appl. Math. **72** (1996), no. 2, 261 – 273.

[ZL99] J. Zhang and Z. Liu, *A Further Study on Inverse Linear Programming Problems*, J. Comput. Appl. Math. **106** (1999), no. 2, 345–359.

[ZZS14] L. Zhang, X. Zhen, and L. Shao, *Learning Object-to-Class Kernels for Scene Classification*, IEEE Trans. Image Process. **23** (2014), no. 8, 3241–3253.