

Modeling and Optimization of Classifiers with Latent Variables

by

Sobhan Naderi Parizi

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Engineering

at the

BROWN UNIVERSITY

© Copyright 2016 by Sobhan Naderi Parizi

This dissertation by Sobhan Naderi Parizi is accepted in its present form by the Department of Engineering as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date.....

Pedro F. Felzenszwalb
Advisor

Recommended to the Graduate Council

Date.....

Stan Sclaroff
Reader

Date.....

Erik Sudderth
Reader

Approved by the Graduate Council

Date.....

Peter M. Weber
Dean of the Graduate School

Acknowledgments

First and foremost, I would like to thank Pedro for advising me during my PhD. He fundamentally transformed the way I think, express my thoughts, implement my ideas, and think about science. I remember times when he would point things out during our meetings, and I would realize the significance of the remarks he casually made weeks later. I would like to thank him for teaching me how to become independent and for his patience with my pace, which in times was very slow.

I would also like to thank my labmates and friends, John Oberlin, Hossein Azizpour, Reza Aghajani, Kun He, Jeroen Chua, and many more wonderful people who were always willing to listen to my crazy thoughts, give me feedback, and help me revise my ideas for the better.

I would also like to thank my family, specially Saman and Mahshid, for their endless support and encouragements during these years.

Last but not least, I would like to thank Beata for making everything so much more fun, and for teaching me new ways of doing things more efficiently.

Abstract of

Modeling and Optimization of Classifiers with Latent Variables

by Sobhan Naderi Parizi, Ph.D.

Brown University, May 2016.

Many applications in Computer Vision and Machine Learning entail learning from partially annotated data. A popular family of models that can capture unobserved variables in the input is the family of latent variable models (LVMs). A big challenge in training LVMs, however, is that their training objective is often highly non-convex leading to problems such as sensitivity to initialization, and convergence to suboptimal solutions. In this thesis, we develop new LVMs for different weakly supervised learning tasks; and propose new optimization frameworks that improve training of latent variable models as well as optimization of other non-convex objectives.

We propose part based models that use latent variables to capture the unknown location of parts in images. We also introduce the notion of “negative parts”, as parts whose detection scores are negatively correlated with the classification score of a class when compared to others. Negative parts are particularly useful in disambiguating similar categories (*e.g.* “cow” and “horse”). We present part based models that can be seen as 3-layer neural networks (ConvNets) with a convolutional layer, followed by a pooling layer, and a fully connected layer. However, we train our models using sequential convex optimization whereas ConvNets use back-propagation for training.

We also introduce a new optimization framework, called Generalized Majorization-Minimization (G-MM), that extends existing approaches to non-convex optimization such as Expectation Maximization (EM) and Concave Convex Procedure (CCCP). EM and CCCP iteratively construct and optimize *tight bounds* on the objective. G-MM, on the contrary, does not require bounds to be tight, making it very flexible. For instance, G-MM can incorporate application-specific biases into the optimization procedure without changing the objective function. We derive G-MM algorithms for several latent variable models and show that they consistently outperform their MM counterparts; in particular, G-MM algorithms are less sensitive to initialization.

Contents

1	Introduction	1
1.1	Classification Problem	3
1.2	Generative Models	4
1.3	Discriminative Models	5
1.3.1	Training and Task Loss	5
1.3.2	Regularization	6
1.4	Support Vector Machines	8
1.5	Training SVMs	9
1.5.1	Gradient Based Approaches	9
1.5.2	Quadratic Programming Solvers	10
1.6	Weakly Supervised Learning	15
1.7	Latent Variable Models	16
1.7.1	Generative Approach	16
1.7.2	Discriminative Approach	17
1.8	Training Latent Structural SVMs	17
2	Reconfigurable Models for Capturing Spatial Layout of Scenes	21
2.1	Bag of Words (BoW) Model	21
2.1.1	Generative Approach	22
2.1.2	Discriminative Approach	23
2.1.3	How Are the Two Approaches Related?	23
2.2	Spatial bag of words (SBoW) Model	24
2.3	Reconfigurable Bag of Words Model	25

2.4	Training Reconfigurable Bag of Words Models	27
2.4.1	Generative Approach	28
2.4.2	Discriminative Approach	30
2.5	Adding Higher-Order Terms	33
2.5.1	Inference with Graph-Cuts	34
2.5.2	Regular Pairwise Functions	35
2.5.3	Reformulating the Regularity Constraints	37
2.6	Experimental Results on Scene Classification	38
2.6.1	15 Scene Dataset	39
2.6.2	MIT 67 Indoor Scenes	39
2.6.3	Synthetic Scene Dataset	42
2.7	Conclusion	47
3	Automatic Part Discovery from Image Collections	49
3.1	Introduction	49
3.2	Part-Based Models for Image Classification	53
3.3	Negative Parts in Binary Classifiers	54
3.4	Negative Parts in Multi-Class Classifiers	55
3.5	Joint Training	56
3.5.1	Caching Hard Examples	58
3.6	Random Part Generation (1 st Step of the Pipeline)	62
3.7	Part Selection (2 nd Step of the Pipeline)	63
3.8	Experiments	64
3.8.1	Visualizing the Model Trained with CNN Features	68
3.8.2	Visualizing the Model Trained with HOG Features	72
3.9	Processing Time	72
3.10	Connection to Convolutional Neural Networks	79
3.11	Conclusion	79
4	Generalized Latent Variable Models	81
4.1	Generalized Latent Variable Models (GLVMs)	81

4.2	Training GLVMs	83
4.2.1	Convex Upper Bound and Concave Lower Bound	84
4.3	Adding Negative Parts to DPMs	85
4.3.1	Training DPMs without Negative Parts	86
4.3.2	Training DPMs with Negative Parts	88
4.3.3	Initializing Negative Parts	90
4.4	Experimental Results	90
4.5	Conclusion	92
5	Generalized Majorization-Minimization	93
5.1	Background	93
5.1.1	Related Work	95
5.2	General Optimization Framework	97
5.2.1	Progress Measure	98
5.2.2	Bound Construction	99
5.3	Convergence Proof of G-MM	101
5.4	Examples of Derived G-MM Algorithms	106
5.4.1	Clustering	107
5.4.2	Detection and Classification with Latent Structural SVM	108
5.5	Experimental Results	110
5.5.1	Clustering with G-MMs	111
5.5.2	Object Detection with G-MMs	113
5.5.3	Image Classification with G-MMs	115
5.5.4	Run Time	119
5.6	G-MM in Expectation (G-MME)	119
5.6.1	Sampling Bounds from Maximum Entropy Distribution	120
5.7	Conclusion	124
6	Conclusion	125

List of Figures

1-1	Visualization of three different per-example losses as a function of the difference between the inference score and the ground-truth score. . .	7
2-1	A Reconfigurable model for a class of outdoor scenes. We have M region models (parts) that can be arranged in different ways to make up an image. Each image region has a preference over the region models that can be used to generate its content. In this example regions in the top are formed by choosing between a cloud or sun region model, while regions in the middle and bottom are formed by choosing between a tree or grass region model.	26
2-2	Three sample images from <i>mountain</i> scene category and their semantic labeling layout. Different instances of a scene category exhibit a large range of variation in their semantic layout.	26
2-3	Some interesting region models learned for different categories using a discriminative RBoW model (Init-EM). Each row illustrates a region model for a particular category. The first column shows the preferences of different image regions for this region model ($A_{y,r,j}$ for fixed y and j). The other columns show image regions that were assigned to this region model during classification ($z_r = j$).	40
2-4	Sample images from the synthetic dataset. Each column shows three images from one scene category. The semantic meaning of each label together with the corresponding observation distribution is shown at the bottom.	44

2-5	Confusion matrices corresponding to different models. The average of the diagonal terms is shown in the parentheses.	46
3-1	Part filters before (left) and after joint training (right) and top scoring detections for each.	50
3-2	Our pipeline. Part selection and joint training are driven by classification loss. Part selection is important because joint training is computationally demanding.	51
3-3	Effect of λ on part norms. Each plot shows sorted ρ_j values for a particular choice of λ	65
3-4	Performance of HOG features on 10-class subset (left) and full MIT-indoor dataset (right).	66
3-5	Performance of CNN features on the full MIT-indoor dataset. <i>HP</i> denotes the hybrid features from [73]. Left: the effect of dimensionality reduction on performance of the CNN features extracted from the entire image. Two approaches are compared; random selection over 5 trials (blue curve) and PCA (red curve). Right: part-based models with random parts (blue curves), selected parts from 1K random parts (red curve), and jointly trained parts (black curve).	68
3-6	Part weights after joint training a model with 52 parts on the full dataset. Patches are represented using 60 PCA coefficients on CNN features. Although the model uses 5 pooling regions (corresponding to cells in $1 \times 1 + 2 \times 2$ grids) here we show the part weights only for the first pooling region corresponding to the entire image.	69
3-7	Top detections of two parts are shown before and after joint training on test images of the full MIT-indoor dataset. The numbers in the first column match the part indices in Figure 3-6.	70

3-8	Top detections of parts on test images of the full dataset. The numbers in the first column match the part indices in Figure 3-6. Part detection is done in a multi-scale sliding window fashion and using a 256×256 window. For visualization purposes images are stretched to have the same size.	74
3-9	Part filters (top) and part weights (bottom) after joint training a model with 52 parts on the 10-class dataset. Here we use HOG features. Although the model uses 5 pooling regions (corresponding to cells in $1 \times 1 + 2 \times 2$ grids) here we show the part weights only for the first pooling region corresponding the entire image.	75
3-10	Top detections of three parts on test images of the 10-class dataset. The numbers in the first column match the part indices in Figure 3-9. Patches from <i>bookstore</i> , <i>laundromat</i> , and <i>library</i> images are highlighted in red, green, and blue respectively (best viewed in color).	76
3-11	Our part-based model can be thought of as a 3-layer Neural Network. The first layer is convolutional. Different channels correspond to different parts. The second layer performs the max-pooling operation in 4 quadrants of the part response maps. The last layer is fully connected and performs 67-way classification.	78
4-1	PR-curve for the <i>cow</i> class. We show result for DPM with 8 parts (blue), GDPM with 6 positive and 2 negative parts initialized (red), and GDPM with 8 positive and 1 negative part initialized by horse DPM model (green).	91
5-1	Optimization of a function F using MM (red) and G-MM (blue). In MM the bound b_2 has to touch F at w_1 . In G-MM we only require that b_2 be below b_1 at w_1 , leading to several choices \mathcal{B}_2	97

5-2	An example of a non-convex function F on \mathbb{R}^d . (a) shows the plot of F ; the value of the function on the line that touches F at the point $p = (0.3, -0.2)$ is marked by the black and white line. (b) shows F from the top view; it also marks the point p with the red circle and shows the direction of the gradient vector at p . The length of the vector is set arbitrarily for visualization purposes. (c) shows the value of F along the tangent line as a one dimensional function (see Equation 5.15); it also shows the gradient direction at $z = 0$ (in red) and the linear function of Equation 5.21 that lower bounds F (in green). . . .	103
5-3	The quadratic function in the left hand side of (5.28). When $\epsilon_1 = \epsilon_2 = 0$, the quadratic function has two roots, $z_1 = 0$ and $z_2 = \frac{A}{2M} > 0$ (shown in blue). When ϵ_1 and ϵ_2 are non-zero, but chosen so that (5.29) holds, the quadratic function is shifted up but still has two distinct roots (shown in red).	107
5-4	The effect of the progress coefficient η (x-axis) on the quality of the solutions found by G-MM on three different clustering datasets. The quality of the solutions is measured by the objective function in (5.30). Lower values are better. The average (solid line), the best (dashed line), and the variance (shaded area) over 50 trials are shown and different initializations are coded with different colors.	113
5-5	Visualization of the solution of k -means and G-MM on the D31 dataset [66] from identical starting point. Random partition initialization scheme is used. (a) color-coded ground-truth clusters. (b) solution of k -means. (c) solution of G-MM. The white crosses indicate location of the cluster centers. Color codes match up to a permutation.	113

5-6	Latent location changes after learning, in relative image coordinates, for all five cross-validation folds, for the <i>top-left</i> initialization on the mammals dataset. Left to right: CCCP, “G-MM random”, “G-MM biased” ($K = 10$). Each cross represents a training image; cross-validation folds are color coded differently. Averaged over five folds, CCCP only alters 2.4% of all latent locations, leading to very bad performance. “G-MM random” and “G-MM biased” alter 86.2% and 93.6% on average, respectively, and perform much better.	115
5-7	Example training images from the mammals dataset, shown with final imputed latent object locations by three algorithms: CCCP (red), G-MM random (blue), G-MM biased (green). Initialization: <i>top-left</i> . . .	116

List of Tables

2.1	Average performance of different methods on the 15 scene dataset. We used three different initialization methods for training a discriminative RBoW model.	39
2.2	Average performance of different methods on the MIT dataset. The last column shows the final value of the LS-SVM objective function for RBoW models with different initializations.	41
2.3	Performance of our reconfigurable model and different baseline methods on the MIT dataset. The last column shows performance of DPM method from [45].	43
4.1	Average Precision (AP) for animal classes of PASCAL VOC 2007 dataset, comparing DPM with GDPM with different number of parts. Sub (super) indices indicate the number of positive (negative) parts.	91
4.2	Cat Head Detection: Results comparing DPM and GDPM with different number of positive and negative parts. Sub (super) indices show the number of positive (negative) parts. GDPM consistently outperforms DPM variants. The average cat head images are taken from [76].	92

5.1	Comparison of G-MM and k -means (hard-EM) on multiple clustering datasets. Three different initialization methods were compared; forgy initializes cluster centers to random examples, random partition assigns each data point to a random cluster center, and k-means++ implements the algorithm from [1]. The mean, standard deviation, and best objective values out of 50 random trials are reported. Both k -means and G-MM use the exact same initialization in each trial. G-MM consistently converges to better solutions.	112
5.2	LS-SVM results on the mammals dataset [27]. We report the mean and standard deviation of the training objective (Equation 5.34) and test error over five folds. Three strategies for initializing latent object locations are tried: image center, top-left corner, and random location. “G-MM random” uses random bounds, and “G-MM bias” uses a bias function inspired by multi-fold MIL [11]. Both variants consistently and significantly outperform the CCCP baseline.	114
5.3	Performance of LS-SVM trained with CCCP and G-MM on MIT-Indoor dataset. We report classification accuracy (Acc.%) and the training objective value (O.F.). Columns correspond to different initialization schemes. “Random” assigns random parts to regions. λ controls the coherency of the initial part assignments: $\lambda = 1$ ($\lambda = 0$) corresponds to the most (the least) coherent case. “G-MM random” uses random bounds and “G-MM biased” uses the bias function of Equation 5.39. $\eta = 0.1$ in all the experiments. Coherent initializations lead to better models in general, but, they require discovering good initial parts. “G-MM” outperforms CCCP, especially with random initialization. “G-MM biased” performs the best. The value of the progress coefficient is set to $\eta = 0.1$ in these experiments.	118

5.4	Comparison of the number of iterations it takes for MM and G-MM to converge in the scene recognition and the data clustering experiment with different initializations and/or datasets. The numbers reported for the clustering experiment are the average and standard deviation over 50 trials.	119
-----	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----

Chapter 1

Introduction

The landscape of machine learning has changed dramatically over recent years. Hundreds of millions of images and millions of hours of video are uploaded onto the Internet daily¹²³. This data is viewed and processed in even larger quantities by Internet users through comments, reviews, view/like-counts, etc. This information is often incomplete, erroneous, ambiguous, and inconsistent. Annotating such a large and ever growing stream of data, on the other hand, is futile. This massive flux of data and lack of annotation creates demand for new applications that create a new set of challenges for machine learning and artificial intelligence.

Traditionally, learning from partially annotated data is solved by weakly and unsupervised training. For example, *Latent Variable Models* (a general family of models that harnesses weakly supervised training) can capture unobserved variables in the input such as the viewpoint of an image, the subcategory of an object, etc. However, the biggest challenge in training such models is that the training objective is often non-convex. This leads to problems such as sensitivity to initialization, and convergence to suboptimal solutions. In this thesis, we (a) propose methods that

¹Based on articles from digitaltrends.com and businessinsider.com, published on 18 September 2013, 350 million photos are uploaded on Facebook daily.

²Based on the information obtained from <https://www.instagram.com/press/> in December 2015, more than 80 million photos are uploaded on instagram.com daily.

³Several sources including reelseo.com and tubefilter.com report that 400 hours of video gets uploaded on Youtube every minute. Also, based on the information obtained from www.youtube.com/press in December 2015, every day hundreds of millions of hours of video is watched on Youtube.

automatically discover visual patterns that are shared across multiple categories using only category-level annotations; (b) extend the family of latent variable models to become more expressive; (c) improve the training of latent variable models.

We propose different generalizations to the existing family of latent variable models and show how different computer vision tasks can benefit from them. We applied these generalizations to solve several computer vision problems including image classification, object detection, and semantic segmentation.

In Chapter 2, we proposed a part based model for scene classification. The model uses latent variables to capture the semantic layout of the scene. So, although the model is trained with only image category labels the model also outputs a semantic segmentation of the input image. In Chapter 3 and 4, we introduce the notion of “negative parts”, parts whose detection scores are negatively correlated with one or more classes. Negative correlations turn out to be important in disambiguating similar categories (*e.g.* “cow” and “horse” in object detection). This can also be thought of as a 2-layer convolutional network that is trained using convex optimization as opposed to back-propagation.

In Chapter 5 we propose new training procedures that extend the existing approaches to non-convex optimization such as expectation maximization (EM) and the Concave Convex Procedure (CCCP). EM and CCCP iteratively construct and optimize convex bounds on the original objective function. The bound at each iteration is required to touch the objective function at the optimizer of the previous bound. In Chapter 5 we show that this touching constraint is unnecessary and overly restrictive. We propose a new optimization framework, Generalized Majorization-Minimization (G-MM), that relaxes this constraint and is much more flexible than CCCP and EM. For instance, G-MM can incorporate application-specific biases into the optimization procedure without changing the objective function. We derive G-MM algorithms for several latent variable models and show that they consistently outperform their MM counterparts in optimizing non-convex objectives. Importantly, in practice, G-MM algorithms are often less sensitive to initialization.

1.1 Classification Problem

In a classification problem the goal is to identify an input as belonging to one of the N pre-defined classes. This is done by a *classifier* which is a function $\hat{y} : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an input $x \in \mathcal{X}$ to one of the N possible categories in $\mathcal{Y} = \{y_1, \dots, y_N\}$.

The classifier \hat{y} may perform the classification task in a number of different ways. For example, *Nearest Neighbor* classifiers, *Neural Networks*, and *Decision Trees* each use a different strategy to perform classification. A common approach, that is used throughout this thesis, involves computing a classification score $f(x, y)$ for each category, and then selecting the highest scoring category, as follows:

$$\hat{y}(x) = \arg \max_{y \in \mathcal{Y}} f(x, y). \quad (1.1)$$

The maximization operation of Equation 1.1, *a.k.a. inference*, is done during test-time. For training, we use a set of labeled training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}$ is the input and $y_i \in \mathcal{Y}$ is the ground-truth output label.

In this thesis, we consider parametric models, meaning that we assume the score function f is parameterized by a set of parameters w . This allows us to formulate the problem of training a classifier as an optimization problem where the goal is to find the model w that optimizes a training objective function $O(w)$ that measures the “goodness” of the classifier defined by w . The objective is typically written as a weighted sum of two terms: 1) a regularization term, denoted by $R(w)$, that is independent of the data and is usually used to penalize complex models, and 2) a data-dependent term, denoted by $\mathcal{L}(\mathcal{D}, w)$, that measures the loss (or the amount of error that the classifier makes) on the training set \mathcal{D} . The loss function is often defined as the sum of losses on training examples $\mathcal{L}(\mathcal{D}, w) = \sum_{i=1}^n L(x_i, y_i, w)$ and the general form of the training objective is as follows:

$$\begin{aligned} O(w) &= R(w) + C\mathcal{L}(\mathcal{D}, w) \\ &= R(w) + C \sum_{i=1}^n L(x_i, y_i, w), \end{aligned} \quad (1.2)$$

where C is a hyper-parameter that balances the regularization penalty and the empirical loss. Finally, the problem of training a classifier can be cast as the problem of finding the minimizer of the training objective, and can be written as follows:

$$w^* = \arg \min_w O(w). \quad (1.3)$$

1.2 Generative Models

Image classification with generative models involves modeling a prior probability over classes, $p(y)$, and the probability of observing certain image features conditional on the image class $p(x | y)$. Using the Bayes law we can classify an image by selecting the class y with the maximum probability given the observed image features:

$$\hat{y}(x) = \arg \max_y p(y | x) = \arg \max_y p(y)p(x | y). \quad (1.4)$$

Note that this becomes the same as Equation 1.1 when we set $f(x, y) = p(y | x)$.

The parameters w of the generative model can be estimated from a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ using a maximum likelihood criteria. If the training examples are independent samples from $p_w(x, y)$, this leads to the following optimization problem:

$$\begin{aligned} w^* &= \arg \max_w \prod_{i=1}^n p_w(y_i)p_w(x_i | y_i) \\ &= \arg \min_w - \sum_{i=1}^n (\log p_w(y_i) + \log p_w(x_i | y_i)). \end{aligned} \quad (1.5)$$

Let $w = \{\gamma, \theta_1, \dots, \theta_N\}$ where γ defines a discrete distribution over categories *i.e.* $p(y) = \gamma_y$ and θ_y defines the parameters of the conditional distribution $p_w(x | y)$ and we have $p_w(x | y) = p_{\theta_y}(x)$. In this case, the maximum likelihood criteria amounts to selecting γ based on the frequency of samples from different classes in the training

set, and θ_y can be estimated using samples from category y alone, as follows:

$$\gamma_y = \sum_{\substack{i=1 \\ y_i=y}}^n \frac{1}{n}, \quad \theta_y = \arg \max_{\theta'_y} \prod_{\substack{i=1 \\ y_i=y}}^n p_{\theta'_y}(x_i).$$

1.3 Discriminative Models

Unlike the generative setting, the discriminative approach does not rely on explicit probabilistic models for the images in each class. Instead the parameters of a classifier are selected to directly minimize mistakes on the training data, often with a regularization bias to avoid overfitting. Motivated by the classification rule in (1.1), a score function $f_w(x, y)$ is trained to score high if image x is from class y , and low otherwise. In particular, we would like to train w such that $f_w(x_i, y_i) > f_w(x_i, y), \forall y \neq y_i$.

Finally we note that one important difference between discriminative and generative training is that discriminative training does not typically decompose into separate training problems for each class. On the other hand, discriminative training methods typically lead to better classification results due to the fact that they directly model the decision boundary, as opposed to explicitly modeling the data.

1.3.1 Training and Task Loss

In this section we focus on the loss term of Equation 1.2. The training loss function $L(x_i, y_i, w)$ should match the test performance measure (*a.k.a.* the target loss). The target loss in a classification problem typically measures the *number* of misclassified training examples. This is called the 0/1 loss and can be written as follows:

$$L_{0/1}(x_i, y_i, w) = \Delta(\hat{y}_w(x_i), y_i) = \mathbf{1}\{\hat{y}_w(x_i) \neq y_i\}. \quad (1.6)$$

The 0/1 loss function is non-differentiable and discontinuous. In particular, the derivative of the 0/1 loss is zero everywhere except for the discontinuity point where it is undefined. Thus, optimizing training objectives that use 0/1 loss is hard, and, in practice, we have to resort to surrogates such as *hinge loss* or *ramp loss*.

Hinge loss is a continuous and convex upper bound to the 0/1 loss. Hinge loss measures the difference between the best loss augmented score (computed via loss augmented inference) and the score under the ground-truth label as follows:

$$L_{\text{hinge}}(x_i, y_i, w) = \max_{y \in \mathcal{Y}} (f_w(x_i, y) + \Delta(y, y_i)) - f_w(x_i, y_i). \quad (1.7)$$

Hinge loss assigns zero penalty to training examples for which the score of the correct (*i.e.* ground-truth) category is higher than the classification score of all *other* categories by a margin of 1. Otherwise, the model gets penalized and the penalty grows linearly with the difference between the score of the correct category and the top-scoring incorrect category. Hinge loss is unbounded from above. This can potentially be problematic in dealing with outliers and/or incorrectly labeled training samples.

Ramp loss is a compromise between 0/1 and hinge loss. It is a continuous non-convex upper bound to the 0/1 loss and can be defined as a truncated hinge loss (as in Equation 1.8) or simply as the difference between the score of loss augmented inference and inference on the input image (as in Equation 1.9).

$$L_{\text{ramp}}(x_i, y_i, w) = \min \left\{ 1, \max_{y \in \mathcal{Y}} (f_w(x_i, y) + \Delta(y, y_i)) - \max_{y \in \mathcal{Y}} f_w(x_i, y) \right\} \quad (1.8)$$

$$= \max_{y \in \mathcal{Y}} (f_w(x_i, y) + \Delta(y, y_i)) - \max_{y \in \mathcal{Y}} f_w(x_i, y) \quad (1.9)$$

Ramp loss is easier than 0/1 loss to work with because it is continuous, but, harder than hinge loss because it is non-convex.

Figure 1-1 illustrates the three loss functions that we discussed in this section.

1.3.2 Regularization

The term $R(w)$ in the training objective of Equation 1.2 is known as the *regularization term*, and is used to control the complexity of the model and to prevent overfitting. ℓ_1 and ℓ_2 regularization are examples of commonly used regularizers and are defined

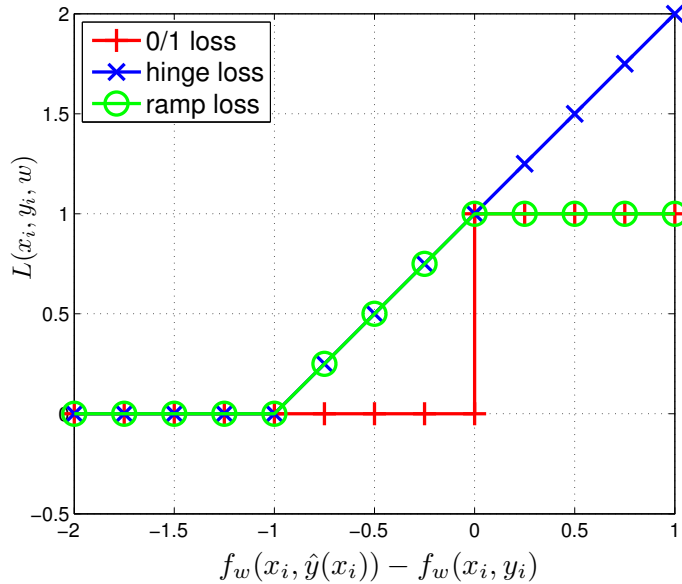


Figure 1-1: Visualization of three different per-example losses as a function of the difference between the inference score and the ground-truth score.

as follows:

$$R_{\ell_1}(w) = \|w\|_1 = \sum_{i=1}^d |w_i|, \quad (1.10)$$

$$R_{\ell_2}(w) = \|w\|_2^2 = \sum_{i=1}^d w_i^2. \quad (1.11)$$

where d is the length of the parameter vector w . R_{ℓ_1} induces sparsity on the learned parameters whereas R_{ℓ_2} leads to dense models. Another popular regularization function is ℓ_1/ℓ_2 (*a.k.a.* group lasso) that can enforce structured sparsity on the trained model. Let $\mathcal{I} = \{1, \dots, |w|\}$ denote an index set. Also, let $I_g \subseteq \mathcal{I}, \forall g \in \{1, \dots, G\}$ denote G different subsets of this index set. Finally, let w_{I_g} denote the dimensions in w that are indexed by I_g . Group lasso enforces structured sparsity patterns according to the grouping defined by I_1, \dots, I_G . It encourages all the parameters within a group (*e.g.* w_{I_g}) to go to zero simultaneously (similar to ℓ_1 regularization) while allowing groups that have non-zero elements to be dense (similar to ℓ_2 regularization). We

denote the regularization function of group lasso by R_g and define it as follows:

$$R_g(w) = \sum_{g=1}^G \|w_{I_g}\|_2. \quad (1.12)$$

1.4 Support Vector Machines

Support Vector Machines (SVMs) [13] are a powerful family of discriminative classifiers that have been extensively used in various problems in Machine Learning and Computer Vision. SVM directly learns the decision boundary that separates training examples of different categories, and when multiple such decision functions are available it chooses the one with the largest margin from examples each category.

SVM was originally designed for binary classification problems where $y_i \in \mathcal{Y} = \{+1, -1\}, \forall i \in \{1, \dots, n\}$. Using the hinge loss and the ℓ_2 regularization function, the training objective of an SVM can be written as follows:

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max\{0, 1 - y_i f_w(x_i)\}, \quad (1.13)$$

where $f_w(x_i)$ denotes distance of sample x_i from the decision boundary.

In many applications the set of possible outcomes has some structure to it. *Structural SVMs (S-SVMs)* are an extended version of support vector machines that are used in problems with complex output structures. For example, in the case of multi-class classification problem with N categories $\mathcal{Y} = \{1, 2, \dots, N\}$, and in the case of part-of-speech tagging \mathcal{Y} depends on the input sentence and comprises of all possible parse trees of the input sentence. The training objective of S-SVMs is as follows:

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \left[\max_{y \in \mathcal{Y}} (f_w(x_i, y) + \Delta(y, y_i)) - f_w(x_i, y_i) \right]. \quad (1.14)$$

The objective functions of SVM (Equation 1.13) and S-SVM (Equation 1.14) are both *strongly* convex and, hence, each has a unique local minimum.

1.5 Training SVMs

Training SVMs requires minimizing the objective function of (1.13) which has a unique local minimum due to its strong convexity property. There are two main strategies for minimizing this objective function. One approach is to treat this as an unconstrained optimization problem and search for the optimal w in the entire space of possible solutions, *i.e.* \mathbb{R}^d . Although the form of the objective function in (1.14) is complicated (it is piecewise quadratic when f_w is linear in w), one can use gradient based methods such as (stochastic) gradient descent [59, 7, 65] to optimize it. Such methods are easy to implement because they only require computing the value of the gradient vector at specific points. Another approach involves turning the complex objective function of (1.14) into a simple (*e.g.* quadratic) optimization problem, as in the cutting-plane method of [30]; but this comes at the price of introducing a large set of constraints to the optimization problem. Fortunately, there are ways to reduce the size of the constraint set and, therefore, make the second approach practical. In some cases, the quadratic problem formulation of SVM training is much faster and easier to work with than the gradient based approaches.

1.5.1 Gradient Based Approaches

Gradient based optimization methods work by taking small steps along the direction of the gradient vector. This boils down to starting from an arbitrary point w , computing (or estimating) the gradient vector and updating w in the appropriate direction accordingly. The strong convexity property of SVM training objective (Equation 1.14) makes it possible to use gradient based methods, such as Stochastic Gradient Descent (SGD) [65] or BFGS [58], to solve the training problem and to converge to the global minimum of the training objective.

For the SVM training objective, each gradient step requires solving inference on *all* training samples and convergence requires numerous gradient steps. This can be prohibitively costly for large datasets. BFGS and SGD use two different strategies to cope with this computational complexity bottleneck. In BFGS, the step sizes are

chosen carefully by taking curvature (the 2nd derivative) of the objective function into account, making it possible to reduce the number of gradient steps needed for convergence by orders of magnitude compared to standard gradient descent. SGD, on the other hand, computes an *approximate* gradient in each step, making gradient computation orders of magnitude faster. In the following section we will describe SGD optimization of SVMs in more details.

1.5.2 Quadratic Programming Solvers

Quadratic Programming (QP) is the problem of optimizing a quadratic function subject to linear (in)equality constraints. Let $w \in \mathbb{R}^d$ denote the vector of variables in the objective function, $b \in \mathbb{R}^m$ denote a vector of constants, and G denote a matrix of size $m \times d$. The general form of a QP with m constraints can be stated as follows:

$$\begin{aligned} w^* &= \arg \min_w F(w) \\ &s.t. \quad Gw \geq b, \end{aligned} \tag{1.15}$$

where F is a quadratic function and each row of G , together with the corresponding element in b , specifies a linear constraint.

We show that when the classification score function is linear in w , that is when $f_w(x, y) = w \cdot \phi(x, y)$, the SVM training objective of (1.13) can be stated as a QP. To this end, we first rewrite the SVM training objective as follows:

$$w^* = \arg \min_w \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max_{(\psi, \delta) \in S_i} w \cdot \psi + \delta, \tag{1.16}$$

where ψ is a vector of feature differences and δ is a 0/1 loss value and S_i is a set of (ψ, δ) pairs associated to the i^{th} training example. Formally, we define S_i as follows:

$$S_i = \{(\psi, \delta) \in \mathbb{R}^d \times \mathbb{R} \mid \psi = \phi(x_i, y) - \phi(x_i, y_i), \delta = \Delta(y, y_i), y \in \mathcal{Y}\}. \tag{1.17}$$

We turn the optimization problem of Equation 1.16 into a quadratic program by

introducing n *slack variables*. Each slack variable, denoted by ξ_i , is associated with the loss value of a training example $(x_i, y_i) \in \mathcal{D}$, and thus, is constrained to be non-negative. The QP equivalent of (1.16) can be written as:

$$\begin{aligned}
 w^* = \arg \min_{w, \xi \geq 0} & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\
 \text{s.t. } & \xi_i - w \cdot \psi \geq \delta, (\psi, \delta) \in S_i, 1 \leq i \leq n.
 \end{aligned}
 \tag{1.18}$$

1-Slack Formulation

One practical problem with the QP equivalent of the SVM training objective that we explained in the previous section is that the number of linear constraints can become large. In particular, the size of the constraint set increases linearly with the size of the training set as well as the number of categories. The issue becomes even more problematic for classifiers with latent variables (which we will discuss in Section 1.7).

Note that the constraints in (1.18) are factorized over training examples, meaning that each training example has its own set of N constraints, denoted by S_i , associated to it and the constraint sets are independent. Each constraint in $(\psi, \delta) \in S_i$ is associated with one of the N possibilities for classification of the i^{th} training example. This results in a total of $n + n \times N$ constraints; n non-negativity constraints for ξ_i s and $n \times N$ constraints for possible interpretations of the training data.

Joachims et al. [30] proposed to define constraints jointly over all training examples. This means that each constraint specifies a possible classification of *all* training examples jointly, therefore, exponentially expanding the size of the constraint set to $n + N^n$. Although, this counterintuitive idea seem to have worsened the issue with the size of the constraint set drastically, it turns out that only a very small fraction of these joint constraints are active at the solution of the optimization problem. More importantly, the number of active joint constraints is affected only by the desired precision of the solution and is *independent* of the number of training examples. We will discuss this in greater detail in the next section.

When working with joint constraints, the QP has only one slack variable which

represents the total sum of the losses of *all* training examples, hence the name “1-slack” formulation. Let $\mathcal{S} = \{e = (e_1, e_2, \dots, e_n) : e_i = (i, \psi, \delta), (\psi, \delta) \in S_i\}$ be the set of all possible joint constraints. In other words, $\mathcal{S} = S_1 \times \dots \times S_n$. Each constraint in \mathcal{S} is an n -tuple whose i^{th} element $e_i = (i, \psi, \delta)$ corresponds to the classification of the i^{th} training example into one of the N possible categories. Note that each joint constraint specifies a *complete* classification of the training set. There is a total training loss value associated to each joint constraint $e = (e_1, \dots, e_n) \in \mathcal{S}$, which we denote by $\text{loss}(e, w)$ and define it as follows:

$$\text{loss}(e, w) = \sum_{i=1}^n w \cdot \psi_i + \delta_i, \quad (1.19)$$

where ψ_i and δ_i are obtained from e_i . The 1-slack formulation of the QP of Equation 1.18 can be written as follows:

$$\begin{aligned} w^* = \arg \min_{w, \xi \geq 0} & \frac{1}{2} \|w\|^2 + C\xi \\ \text{s.t. } & \text{loss}(e, w) \leq \xi, \forall e \in \mathcal{S}. \end{aligned} \quad (1.20)$$

Maintaining a Working Set of Active Constraints

In practice, we cannot optimize (1.20) since the size of the constraint set \mathcal{S} is too large. However, it turns out that the dual of the QP in (1.20) has an extremely sparse solution in the sense that most of the dual variables turn out to be zero [30]. Since dual variables correspond to constraints in the primal form the observation implies that only a tiny fraction of the primal constraints will be *active*. This is the key behind conception of an efficient algorithm, which we will explain in the rest of this section, that can solve the QP formulation of SVM training problem in practice.

As we discussed, almost all of the constraints in (1.20) will be “inactive” at the solution of the optimization problem except for a manageable small subset, $W \subseteq \mathcal{S}$. This observation suggests that one can take an iterative approach to solve the optimization problem of (1.20) in practice. In each round we alternatively optimize the quadratic objective subject to the constraints in W and add some of the violated

constraints from $\mathcal{S} \setminus W$ to W . This is also known as the *cutting-plane method* in optimization literature. This iterative process is repeated until the algorithm converges to the solution of the original optimization problem. Convergence is declared if the gap between the value of the dual and the primal objective at the current solution gets smaller than a desired tolerance ϵ . Joachims et al. [30] showed that if the *most* violated constraint⁴ is added to W in each round the number of steps that the algorithm requires to converge is linear in the desired precision of the solution. In particular, the number of iterations is inversely proportional to ϵ and does *not* depend on the size of the training set. This leads to an optimization algorithm for training SVMs that is linear in the number of training examples, because finding the most violated constraint in each round requires doing inference on all the training examples.

Dual Formulation

Motivated by the observation made in [30] regarding the extreme sparsity of the solution of the dual of (1.20), it may be faster to work with the dual form.

Let $\Psi(e) = \sum_{i=1}^n \psi_i$ where $e = (e_1, \dots, e_n) \in W$ and $e_i = (i, \psi_i, \delta_i)$. Also, let $e^{(j)}$ denote the j -th element in W (here we are assuming W is an ordered set). We define M to be a $|W| \times |W|$ kernel matrix for the feature function Ψ . So, we have $M_{j,k} = \Psi(e^{(j)}) \cdot \Psi(e^{(k)})$. Also, let b be a vector of length $|W|$ where for any $e \in W$, b_e is defined as $b_e = -\sum_{i=1}^n \delta_i$. The dual of (1.20) subject to constraints in $W \subseteq \mathcal{S}$ can be written in the following simple form:

$$\begin{aligned} \alpha^* &= \arg \min_{\alpha \geq 0} \frac{1}{2} \alpha^T M \alpha + \alpha^T b \\ \text{s.t. } & \sum_{e \in W} \alpha_e \leq C, \end{aligned} \tag{1.21}$$

where $\alpha = (\alpha_1, \dots, \alpha_{|W|})$ is the vector of dual variables. The solution of the dual and

⁴The most violated constraint is the one with the highest loss value i.e. $\arg \max_{e \in \mathcal{S}} \text{loss}(e, w)$.

Algorithm 1 Fast QP solver for optimizing SVM training objective.

Input: convergence precision ϵ

```

1:  $W := \emptyset$ 
2: repeat
3:    $M[e, e'] := \Psi(e) \cdot \Psi(e'), \forall e, e' \in W$ 
4:    $b[e] := -\sum_{i=1}^n \delta_i, \forall e \in W$   $\triangleright e = (e_1, \dots, e_n), e_i = (i, \psi_i, \delta_i)$ 
5:    $\alpha^*$  := the solution of the QP (Equation 1.21)
6:    $w^* := -\sum_{e \in W} \alpha_e^* \Psi(e)$ 
7:   prune  $W$ 
8:    $e^* := \arg \max_{e \in W} \text{loss}(e, w^*)$   $\triangleright$  most violated constraint
9:    $W := W \cup \{e^*\}$ 
10:   $\xi^* := \frac{-1}{C}(\alpha^{*T} M \alpha^* + \alpha^{*T} b)$ 
11: until  $\text{loss}(e^*, w) \leq \xi^* + \epsilon$ 

```

Output: w

the primal are related through the following equation:

$$w^* = -\sum_{e \in W} \alpha_e^* \Psi(e). \quad (1.22)$$

We start from an empty set of constraints $W = \emptyset$ and gradually add constraints to W . After enough iterations, many of the α_e 's become (and remain) zero for the rest of the optimization process. This happens in particular for the constraints that were added in the earlier rounds of growing W . This observation suggests that we can prune the constraint set W in each iteration by discarding e 's for which α_e has remained zero for a certain number of consecutive iterations. We use Algorithm 1 to solve the optimization problem of Equation 1.20 in practice. Line 10 of the algorithm computes the value of the slack variable of the primal form (Equation 1.20) by setting the value of the solution of the primal to that of the dual. The duality gap is zero because the optimization problem is quadratic and convex (the matrix M is positive semi definite). Note, however, that the dual problem was originally a maximization problem and we stealthily changed it to a minimization problem by negating the objective function. Thus, the solution of (1.20) is equal to the negative of the solution of (1.21) and ξ

can be computed as follows:

$$\frac{1}{2}\|w^*\|^2 + C\xi^* = -\frac{1}{2}\alpha^{*T}M\alpha^* - \alpha^{*T}b, \quad \alpha^{*T}M\alpha^* = w^{*2} \quad (1.23)$$

$$\Rightarrow \xi^* = \frac{-1}{C}(\alpha^{*T}M\alpha^* + \alpha^{*T}b). \quad (1.24)$$

Improving Runtime using Sub-Optimal Cutting-Planes

Finding the most violated constraint (line 8 in Algorithm 1) is linear in the number of training examples. This makes the algorithm slow for training SVMs on very large training sets. Finding *the most* violated constraint requires running loss augmented inference on all training examples. The training algorithm, however, should still make improvement if any violating constraint is added to the working set W . Although adding sub-optimal cutting-planes (or constraints) delays the convergence of the algorithm in terms of the number of iteration, but, it can significantly reduce the time spent on updating W . We observed up to 4x speedup in total training time by running loss augmented inference on only a fraction of the training data when updating W . When finding a new cutting-plane, we randomly choose 20% of the training data and only update the loss augmented inference result for those selected examples. For all the other training examples we use the classification result of the last cutting-plane. We reserve a small chance (5%) to do a normal full update (*i.e.* run loss augmented inference on all training examples) in each round.

1.6 Weakly Supervised Learning

Many interesting models capture aspects of the data that is not labeled in the training data. For example, consider a scene classification problem where we would like the classifier to determine the class based on an intermediate representation of the input for which we do not have annotation information. The intermediate representation can be a set of attributes (*e.g.* indoor, outdoor, natural, man-made, open area, round, rugged, etc.) or the output of a set of object detector (*e.g.* has human, has car, has chair, has animal, etc.). We can design the model so that it has the ability to

capture such information and yet train it only with the class-level annotation. This is called *weakly supervised* learning problem. Often the category label annotation provides enough signal to drive the weakly supervised learning towards discovering good intermediate representations.

1.7 Latent Variable Models

Output variables are those variables in the model that denote the prediction of the model. The ground-truth value of the output variables is typically known for all the examples in the training set. *Latent variables* can be used to capture unobserved structures present in the data; that is the structures for which we do not have access to the ground-truth values during training. Latent variables are usually treated as auxiliary variables that are either *marginalized* out (Equation 1.25a) or *maximized* over (Equation 1.25b) during inference.

$$f_w(x, y) = \begin{cases} \sum_z f_w(x, y, z) & \text{(marginalized)} & (1.25a) \\ \max_z f_w(x, y, z) & \text{(maximized)} & (1.25b) \end{cases}$$

1.7.1 Generative Approach

To augment generative models with latent variable we define $p(x | y)$ in terms of a distribution $p(z | y)$ over latent values conditional on the class of the image, and the probability of observing certain image features conditional both on the image class and the latent values $p(x | z, y)$. Then $p(x | y)$ is obtained by integrating over the latent variables, similar to Equation 1.25a,

$$p(x | y) = \sum_z p(x | z, y)p(z | y). \quad (1.26)$$

Maximum likelihood parameter estimation (Equation 1.5) with latent variable models typically leads to non-convex optimization problems. The Expectation-Maximization (EM) algorithm is a general tool for dealing with such problems.

1.7.2 Discriminative Approach

We focus on score functions that are linear in w given the output structure y and the latent configuration z , that is $f_w(x, y, z) = w \cdot \phi(x, y, z)$. Moreover, we consider the case where the latent variables are maximized over. Such models have been successfully used in many vision applications such as [20, 47, 62]. Let $Z(x)$ denote the space of all possible latent configurations for the input x . We can write the classification score function and the inference rule as follows:

$$f_w(x, y) = \max_{z \in Z(x)} w \cdot \phi(x, y, z), \quad (1.27)$$

$$\hat{y}(x) = \arg \max_{y \in \mathcal{Y}} f_w(x, y) = \arg \max_{y \in \mathcal{Y}} \max_{z \in Z(x)} w \cdot \phi(x, y, z). \quad (1.28)$$

We can replace this in the SVM training objective of (1.14) to get the training objective for *Latent Structural SVM (LS-SVM)* [20, 71], described in Equation 1.29. A popular example in computer vision is the deformable part model (DPM) for object detection described in [20]. The work in [20] considered the special case of a latent variable binary classifier (the object is present or not at each position in the image).

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \left[\max_{\substack{y \in \mathcal{Y} \\ z \in Z(x_i)}} (w \cdot \phi(x_i, y, z) + \Delta(y, y_i)) - \max_{z \in Z(x_i)} w \cdot \phi(x_i, y_i, z) \right] \quad (1.29)$$

1.8 Training Latent Structural SVMs

Unfortunately, the LS-SVM training objective in (1.29) is non-convex. This is due to the last max operation in Equation 1.29. In [71], the LS-SVM training objective is optimized using the *Concave-Convex Procedure (CCCP)* algorithm [72], while [20] uses a coordinate descent method designed for the binary case. While these methods have been shown to work well in some applications there is increasing evidence that they can be quite sensitive to initialization. Our experiments confirm this is a significant problem for various latent variable models. In contrast, the EM algorithm for generative models with latent variables seems to be less sensitive to initialization.

CCCP provides an algorithm for minimizing objective functions that can be written as sum of a convex and a concave function; or, equivalently, difference of two convex functions. CCCP alternates between replacing the concave part with a linear function that is tangent to the objective function at the current solution and optimizing the resulting function. The resulting function is convex, upper bounds the true objective function, and touches it at the current solution. These three properties guarantee that the CCCP algorithm makes progress in each round and converges to an extremum or a saddle point of the original (non-convex) objective function.

The concave part of Equation 1.29 can be linearized by fixing the latent variables in the last term in the equation. In order to ensure that the resulting function touches the original function at the solution of the t -th iteration (denoted by $w^{(t)}$), we fix the latent variables as follows:

$$z_i(w^{(t)}) = \arg \max_{z \in Z(x_i)} w^{(t)} \cdot \phi(x_i, y_i, z). \quad (1.30)$$

Let $\mathbf{z}^{(t)} = (z_1(w^{(t)}), \dots, z_n(w^{(t)}))$ denote the fixed values of the latent variables for all training examples. We denote the upper bound that CCCP uses in iteration t by $B(w; \mathbf{z}^{(t)})$ and define it in (1.31).

$$B(w; \mathbf{z}^{(t)}) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \left[\max_{\substack{y \in \mathcal{Y} \\ z \in Z(x_i)}} (w \cdot \phi(x_i, y, z) + \Delta(y, y_i)) - w \cdot \phi(x_i, y_i, \mathbf{z}_i^{(t)}) \right] \quad (1.31)$$

Note that $O(w^{(t)}) = B(w^{(t)}; \mathbf{z}^{(t)})$, and therefore, B touches O at $w^{(t)}$.

Algorithm 2 explains the CCCP algorithm for training LS-SVMs. The algorithm alternates between *relabeling* the latent variables of all training examples to construct a new convex bound (line 6) and *updating* the model to minimize the bound (line 7).

The algorithm converges when the update step does not change the latent values; that is when $\mathbf{z}^{(t+1)} = \mathbf{z}^{(t)}$. In practice, however, convergence is declared when $O(w^{(t)})$ changes by less than ϵ in several (*e.g.* 5) consecutive iterations.

The bound function B of (1.31) is convex, and hence, can be minimized using gradient based approaches, including the ones discussed in Section 1.5.1. Also, the

Algorithm 2 CCCP algorithm for training Latent Structural SVMs.

Input: initial model $w^{(0)}$ or initial fixed latent variables $\mathbf{z}^{(0)} = (z_1, \dots, z_n)$

- 1: **if** initial latent variables (z_1, \dots, z_n) are provided **then**
- 2: $w^{(0)} := \arg \min_w B(w; \mathbf{z}^{(0)})$, from (1.31)
- 3: **end if**
- 4: $t := 0$
- 5: **repeat**
- 6: $\mathbf{z}^{(t+1)} := (z_1(w^{(t)}), \dots, z_n(w^{(t)}))$, from (1.30) ▷ Relabeling step
- 7: $w^{(t+1)} := \arg \min_w B(w; \mathbf{z}^{(t+1)})$, from (1.31) ▷ Update step
- 8: $t := t + 1$
- 9: **until** convergence

Output: $w^{(t)}$

bound can be written in the form of the objective function of (1.16), and hence, can be minimized using QP solvers and the techniques that we discussed in Section 1.5.2. Recall the observation regarding the extreme sparsity of the solution of the QP in (1.21). This may become particularly useful for training LS-SVMs because the maximization over the latent variables in B (Equation 1.31) may make the loss augmented inference very costly, making gradient based methods prohibitively slow. Sparsity of the solution of the QP suggests that the QP solver method of Algorithm 1 may take only a modest number of iterations to converge.

Chapter 2

Reconfigurable Models for Capturing Spatial Layout of Scenes

In this chapter we propose a new latent variable model for image classification and scene recognition. Our approach represents a scene as a collection of region models (“parts”) arranged in a reconfigurable pattern. We partition an image into a pre-defined set of regions and use a latent variable to specify which part is assigned to each image region. We use a bag of words representation to capture the appearance of an image region. The resulting method generalizes a spatial bag of words approach that relies on a fixed model for the bag of words in each image region.

Our models can be trained using both generative and discriminative methods. In the generative setting we use the Expectation-Maximization (EM) algorithm to estimate model parameters from a collection of images with category labels. In the discriminative setting we use a Latent Structural SVM (LS-SVM). We note that LS-SVMs can be very sensitive to initialization and demonstrate that generative training with EM provides a good initialization for discriminative training with LS-SVM.

2.1 Bag of Words (BoW) Model

A bag of words (BoW) model represents an image x by an unordered collection of visual words. Suppose we have a dictionary with K visual words. A bag of words b

is defined by a vector $[b_1, \dots, b_K]$ where b_k is the multiplicity of word k in b . We use $|b| = \sum_{k=1}^K b_k$ to denote the total number of words in b .

2.1.1 Generative Approach

A generative BoW classifier assumes that the visual words in an image are independent samples from a multinomial distribution conditioned on the image class.

A multinomial distribution is defined by a discrete distribution with parameters $v = \{v_1, \dots, v_K\}$ specifying the probability of each outcome in a trial. In the multinomial model each word in a bag is generated independently. The probability of a bag b (conditional on $|b|$) is given by

$$\text{mult}(b, v) = \frac{|b|!}{b_1! \dots b_K!} \prod_{k=1}^K v_k^{b_k}. \quad (2.1)$$

To define a BoW classifier, let θ_y specify a discrete distribution over visual words associated with class y . Then

$$p_\theta(x|y) = \text{mult}(x, \theta_y). \quad (2.2)$$

We can estimate the model parameters θ from a set of training examples using a maximum likelihood criteria. The parameters θ_y simply depend on the frequencies of different visual words observed in images from class y . If we assume the training examples in $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ were generated independently from $p(x, y)$ the maximum likelihood parameters are given by

$$\theta_{y,k} = \frac{c_{y,k}}{\sum_{k'=1}^K c_{y,k'}}, \quad \text{and } c_{y,k} = \sum_{\substack{i=1 \\ y_i=y}}^n x_{i,k}, \quad (2.3)$$

where $c_{y,k}$ is the number of times the k^{th} visual word was seen in images from class y .

To classify an image we combine the generative model associated with each class $p(x|y)$ with the prior probability of each class $p(y)$ as specified by Equation (1.4).

2.1.2 Discriminative Approach

We define a discriminative BoW classifier using a discriminant function of the form

$$f_w(x, y) = w_y \cdot \phi(x). \quad (2.4)$$

Here $w = [w_1; \dots; w_N]$ denotes a vector of model parameters where w_y are parameters associated with class y . The function $\phi(x)$ is a (possibly non-linear) feature map of the bag of words in image x .

Since $f_w(x, y)$ is linear in w this leads to a Structural SVM and we can train w by minimizing the objective function of (1.14). As mentioned before, the optimization problem defined by Structural SVMs is convex and can be solved using a variety of techniques, including stochastic gradient descent (Section 1.5.1) and quadratic programming (section 1.5.2).

2.1.3 How Are the Two Approaches Related?

We note that the classification rule obtained with the generative BoW model with parameters θ can be written similar to that of a discriminative model as follows:

$$\hat{y}(x) = \arg \max_y w_y \cdot \phi(x) \quad (2.5)$$

$$\phi(x) = [\psi(x); 1], \quad \psi(x) \text{ is the BoW representation of } x, \quad (2.6)$$

$$w_y = [\log \theta_{y,1}; \dots; \log \theta_{y,K}; \log \gamma_y], \quad \gamma_y = p_\theta(y). \quad (2.7)$$

Therefore, the discriminative model can be seen as “more general” than the generative model. In particular, the classification rule defined by the generative model is in the space of classification rules considered by discriminative training if we choose $\phi(x)$ properly. Discriminative training is more agnostic in the sense that it selects a classification rule by directly minimizing a regularized risk on the training set, instead of making assumptions about how the data was generated. Moreover, we can train a discriminative classifier using any arbitrary feature map $\phi(x)$.

2.2 Spatial bag of words (SBoW) Model

Following [35] we can take spatial information into account by using a different model for the features in different regions of an image. This leads to *Spatial BoW (SBoW)* models. Here we consider the case where the image is partitioned into a fixed grid with R regions. We use r to denote an image region and x_r to denote the bag of words in region r of an image x .

In a generative SBoW model we capture spatial information by allowing the probability of observing a particular visual word to depend on the region where the word is observed. Let $\theta_{y,r}$ denote a discrete distribution over visual words associated with region r and class y . Under the SBoW model we have

$$p_{\theta}(x|y) = \prod_{r=1}^R \text{mult}(x_r, \theta_{y,r}). \quad (2.8)$$

As in the case of a BoW model we can estimate the model parameters from a set of training examples using a maximum likelihood criteria. The parameters $\theta_{y,r}$ simply depend on the frequencies of different visual words observed in region r taken over images in class y and $\theta_{y,r,k}$ is estimated similar to Equation 2.3.

We define a discriminative SBoW model using a discriminant function of the form

$$f_w(x, y) = w_y \cdot [\psi(x_1); \dots; \psi(x_R); 1]. \quad (2.9)$$

As in the discriminative BoW model, w_y denotes the parameters of class y , but now w_y has different parameters for modeling the visual words in each image region; *i.e.* $w_y = [w_{y,1}; \dots; w_{y,R}; w_{y,b}]$, where $w_{y,b}$ denotes the bias term associated with category y . Since f_w is still linear in w , we can once again train w using a S-SVM.

Like in the case of BoW models, the discriminative SBoW model is “more general” than the generative SBoW model because it can represent the same decision functions while being more agnostic and allows for using any feature map.

2.3 Reconfigurable Bag of Words Model

Consider an image of a beach scene. We expect to see sky, water and sand in the image. Moreover, we expect to see sky at the top of the image, water somewhere in the middle and sand in the bottom. One approach for capturing this information involves using a different bag of words model for different regions in the image. This structure can be modeled by spatial pyramid matching [35]. Note however that a region in the middle of the image could contain water or sand. Similarly a region at the top of the image could contain a cloud, the sun or blue sky alone. Therefore the features observed in each region depend on a latent variable specifying which of several possible region models should be used to capture the content of the region.

We propose to model a scene as a collection of region models (“parts”) arranged in a reconfigurable pattern. An image is divided into a set of pre-defined regions and we have latent variables specifying which region model should be used for each image region. The model includes parameters so that each image region has a preference over the region models that can be assigned to it. In practice we divide an image into a grid of regions and use a bag of words (BoW) representation to capture the appearance of a region. We call the resulting models *Reconfigurable BoW* models.

Figure 2-1 illustrates a model for a class of outdoor scenes composed of sky, grass and trees. We can think of the model as being defined by parts that model image regions with specific content. The latent variables specify which part should be used to capture the appearance of each region in the grid. Figure 2-2 demonstrates three sample images from the same scene category along with the color-coded ground-truth part assignments of image regions. The figure shows the extent of reconfiguration in scene layouts.

Related Work

We compare reconfigurable BoW models to spatial BoW models that use a fixed model for the bag of words in each image region and show that reconfigurable models lead to superior results on two datasets.

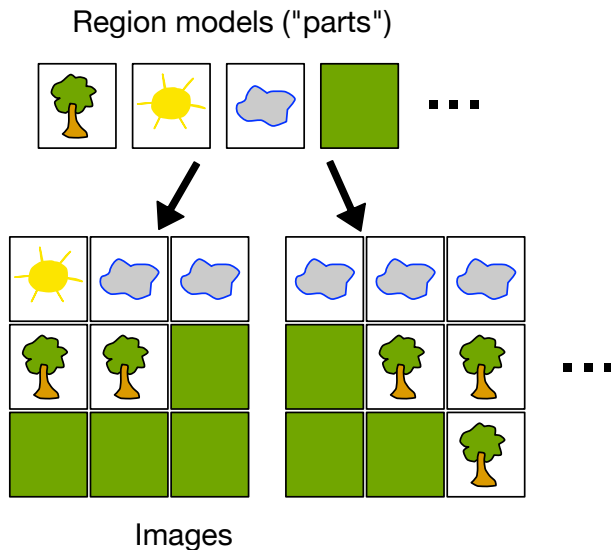


Figure 2-1: A Reconfigurable model for a class of outdoor scenes. We have M region models (parts) that can be arranged in different ways to make up an image. Each image region has a preference over the region models that can be used to generate its content. In this example regions in the top are formed by choosing between a cloud or sun region model, while regions in the middle and bottom are formed by choosing between a tree or grass region model.

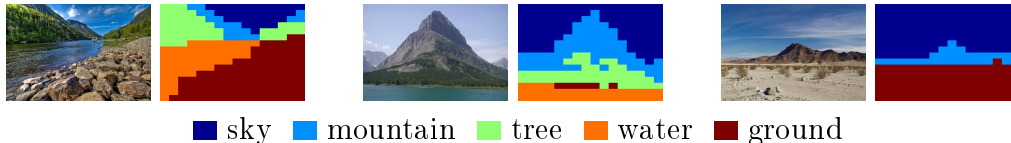


Figure 2-2: Three sample images from *mountain* scene category and their semantic labeling layout. Different instances of a scene category exhibit a large range of variation in their semantic layout.

The idea of modeling a scene in terms of a configuration of regions with specific properties goes back to the work in [39]. This notion has also been used recently for recognizing indoor scenes in [52] and [45]. These methods represent scenes using different kinds of deformable part models. Reconfigurable models are different from deformable models because they explicitly model the whole image. Reconfigurable models also allow the same part (region model) to be used multiple times in an image. For example, a grass region model can be instantiated at multiple locations to explain a large patch of grass in an image.

Latent variable models have become very popular in the context of object detec-

tion following the work in [20]. One of our goals is to understand how to define latent variable models for scene recognition. [45] used the deformable part models (DPM) from [20] and obtained good scene recognition results after combining DPMs with several global image descriptors. Our latent variable models can be seen as an alternative to DPMs that is more closely related to methods based on global descriptors.

Another kind of latent variable model that has been used for scene recognition involves hierarchical “topic” models [19, 6]. These models represent the features in an image using a mixture of topics associated with an image category. They are related to Reconfigurable models if we think of a region model as a topic. In the case of a reconfigurable model we assume there is a single topic in each image region. Here we train different region models for each image category but we could also share a set of region models over all categories as is often done with topic models.

The approach in [69] is closely related to ours from a technical point of view but they use only two region models for “foreground” and “background” regions while we use many different region models to model a scene.

Recently, *tangram* models [75] and *HST* (*hierarchical space tiling*) models [67] were proposed that offer a compact and reconfigurable representation for scene categories. These models use training data to learn a mixture of templates for each category. A template is defined in terms of a tiling of the image and an assignment of a feature type (*e.g.* texture, color, etc.) to each tile. Although tangram models provide some degree of reconfigurability by learning a few templates for each category, however, we believe that it is not enough to capture all possible variations in the layout of a scene. We believe that reconfiguration should be considered in the instance (image) level.

2.4 Training Reconfigurable Bag of Words Models

Our latent variable model builds on the SBoW model. We can think of an SBoW model as a part-based approach with one part per image region. Here we augment the SBoW model to allow for reconfiguration of the parts that make up a scene. This leads to a class of *reconfigurable bag of words* (*RBoW*) models.

For example, an RBoW model for beach scenes might have a part modeling an image region that contains the sun and another part modeling an image region that contains a cloud. The regions at the top of a beach image could all contain the sun or a cloud. In the RBoW model we have a latent variable indicating which region model (part) should be used to explain each image region.

As in an SBoW model we assume images are partitioned into R predefined regions and x_r specifies the bag of words observed in region r within the image x . In a reconfigurable BoW model we have M BoW region models. A latent value z_r assigns a particular region model to region r .

We describe both generative and discriminative version of the reconfigurable BoW model. For the generative models we use Expectation-Maximization (EM) [5] to train model parameters. For the discriminative models we use a Latent Structural SVM (LS-SVM) [71]. Discriminative training usually outperforms generative training but we have found that LS-SVM training is much more sensitive to initialization when compared to EM training. We show that a combined approach that initializes LS-SVM training using the results of EM training gives the best performance.

2.4.1 Generative Approach

In the RBoW model we assume the visual words in image region r are generated independently conditional on the class label and latent value z_r assigning a region model to region r .

Let $W_{y,j}$ be a discrete distribution over visual words associated with the j -th region model for class y . Under the RBoW model we have

$$p_{\theta}(x|z, y) = \prod_{r=1}^R \text{mult}(x_r, W_{y,z_r}) \quad (2.10)$$

We assume the latent values z_r are independent conditional on the class label y but not identically distributed. There is a different categorical distribution capturing which parts are likely to occur in each region of an image from a particular class. For each class y and region r let $a_{y,r} = \{a_{y,r,1}, \dots, a_{y,r,M}\}$ where $a_{y,r,j}$ is the probability

that $z_r = j$ on an image from class y . This leads to the following distribution over the latent values

$$p_\theta(z|y) = \prod_{r=1}^R a_{y,r,z_r}. \quad (2.11)$$

Now we can express the probability of observing the features in an image x conditional on an image class y by integrating over the possible latent values

$$p_\theta(x|y) = \sum_z p_\theta(x|z, y) p_\theta(z|y). \quad (2.12)$$

Since the latent values are independent and the observations are independent conditional on the latent values we can compute this probability efficiently (in $O(RLK)$ time for a model with M region models on an image with R regions and a dictionary with K visual words) as

$$p_\theta(x|y) = \prod_{r=1}^R \sum_{z_r} \text{mult}(x_r, W_{y,z_r}) a_{y,r,z_r}. \quad (2.13)$$

The parameters θ_y associated with the model for class y are given by M BoW region models $\{W_{y,1}, \dots, W_{y,M}\}$ and R distributions over region models $\{a_{y,1}, \dots, a_{y,R}\}$.

Parameter estimation with EM Suppose we have n training examples $\{(x_1, y_1), \dots, (x_n, y_n)\}$. We can estimate the parameters of an RBoW model using a maximum likelihood criteria, but since the model has latent variables maximum likelihood estimation leads to a non-convex optimization problem. We use the Expectation-Maximization (EM) algorithm to address this problem [5].

EM computes a sequence of model parameters by repeatedly alternating between two steps which are guaranteed to increase the likelihood of the data. In the E step we use the current model θ to compute the posterior probability of the latent variables in each training example. This gives us a tractable lower-bound on the likelihood function which is tangent to the actual likelihood at the current θ . In the M-step we update the model parameters by maximizing the lower-bound on the likelihood

function. In the case of an RBoW model we obtain the following algorithm.

Repeat until convergence

Step 1 (E): For each example i , region r and latent value z_r compute $Q_{i,r,z_r} = p_\theta(z_r|x_i, y_i)$ using

$$p_\theta(z_r|x_i, y_i) = \frac{\text{mult}(x_r, W_{y,z_r})a_{y,r,z_r}}{\sum_j \text{mult}(x_r, W_{y,j})a_{y,r,j}} \quad (2.14)$$

Step 2 (M): Update θ by selecting

$$a_{y,r,j} \propto \sum_{i,y_i=y} Q_{i,r,j} \quad (2.15)$$

$$W_{y,j,k} \propto \sum_{i,y_i=y} \sum_r Q_{i,r,j} c_{i,r,k}, \quad (2.16)$$

where $c_{i,r,k}$ is the number of times the k -th visual word was seen in region r of x_i and the parameters are normalized so that $\sum_j a_{y,r,j} = 1$ and $\sum_k W_{y,j,k} = 1$.

We initialize the algorithm by selecting a random latent value z_r for each region r within x_i and setting $Q_{i,r,z_r} = 1$ while $Q_{i,r,j} = 0$ for $j \neq z_r$.

In practice we smooth the multinomial probabilities in (2.14) by raising them to a power of $1/T$. This attenuates the sharpness induced by the assumption that visual words are generated independently within a region. This becomes particularly important when using densely sampled features.

2.4.2 Discriminative Approach

We define a discriminative RBoW classifier using a discriminant function of the form

$$f_w(x, y) = \max_z \sum_r A_{y,r,z_r} + B_{y,z_r} \cdot \psi(x_r). \quad (2.17)$$

$\psi(x_r)$ is a feature map that computes a representation (*e.g.* BoW) of the information in region r of the image (x_r). The vector $B_{y,j}$ specifies model parameters for the j -th region model in class y . The parameter $A_{y,r,j}$ specifies a score for assigning part

j to region r in an image of class y . Intuitively, for each class y , we attempt to explain the image x by finding the best assignment z of parts to the regions in x . Score of assignment z comprises of two terms. $\sum_r A_{y,r,z_r}$ computes a prior score that is independent of the image data. $\sum_r B_{y,z_r} \cdot \psi(x_r)$ is a data dependent term that measures, for each region r , how well part z_r explains the content of x_r .

Latent Structural SVM Let w_y denote the concatenation of the parameters A_{y,r,z_r} and B_{y,z_r} . Also, let $\Phi(x, z)$ be a sum of R vectors $\Psi(x, r, z_r)$, one per image region. The vector $\Psi(x, r, z_r)$ equals $\psi(x_r)$ in the dimensions corresponding to B_{y,z_r} within w_y and 1 in the dimension corresponding to A_{y,r,z_r} within w_y . The other entries in $\Psi(x, r, z_r)$ are zero. The classification score function of (2.17) can be expressed simply as a dot product of the following form:

$$f_w(x, y) = \max_z w_y \cdot \Phi(x, z). \quad (2.18)$$

Let $w = [w_1; \dots; w_N]$ denote a vector with all model parameters from all classes. Suppose we have n training examples $\{(x_1, y_1), \dots, (x_n, y_n)\}$. We can train a discriminative RBoW model by minimizing the Latent Structural SVM training objective as follows (see Section 1.8 for more details):

$$w^* = \arg \min_w \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max_{y,z} (w_y \cdot \Phi(x_i, z) + \Delta(y, y_i)) - \max_z w_{y_i} \cdot \Phi(x_i, z). \quad (2.19)$$

Similar to a Structural SVM, in a Latent Structural SVM the training objective encourages the score of the correct class to be above the highest score of an incorrect class by a margin of one. The only difference is that the score is no longer linear, and instead involves a maximization over z .

Unfortunately the optimization problem defined by an LS-SVM is not convex, but, we can use Algorithm 2 from Chapter 1 to find a solution that is locally optimum. The algorithm implements the CCCP method of [72] by repeatedly alternating between two steps. The first step picks the best latent values for each training example under the current model $w^{(t)}$. We denote the fixed latent values for the i -th training example

by $z_i(w^{(t)})$ and compute them as follows:

$$z_i(w^{(t)}) = \arg \max_z w_{y_i} \cdot \Phi(x_i, z).$$

The second step defines a convex upper bound on the training objective and minimizes the bound. The bound is denoted by $B(w; \mathbf{z}^{(t)})$ where $\mathbf{z}^{(t)} = (z_1(w^{(t)}), \dots, z_n(w^{(t)}))$. For the training objective of (2.19) the bound is defined as follows:

$$B(w; \mathbf{z}^{(t)}) = \arg \min_w \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max_{y, z} (w_y \cdot \Phi(x_i, z) + \Delta(y, y_i)) - w_{y_i} \cdot \Phi(x_i, z_i(w^{(t)})).$$

$B(w; \mathbf{z}^{(t)})$ is convex and can be minimized using standard approaches such as gradient based methods (see Section 1.5.1) or quadratic programming (see Section 1.5.2).

The CCCP optimization for LS-SVM is similar to EM in the way that it alternates estimating latent values and estimating model parameters. One important difference is that in step 1 of EM we obtain a distribution over latent values for each example while here we pick a single latent value for each example. This seems to make LS-SVM optimization with CCCP much more sensitive to initialization. In practice we find that most imputed latent values, *i.e.* the elements of $\mathbf{z}^{(t)}$, never change.

The optimization requires either an initial weight vector w or initial latent values z_i , in which case training starts in step 2. We experimented with three different methods for selecting initial latent values. One method simply picks a random region model for each region in each image. Another method picks a particular region model for each region. In particular, we train models with 16 regions and 16 region models and assign a different initial region model for each image region. Finally, we tried using the result of EM training of a generative RBoW model to select the initial latent values. In this case we set the initial z_i to be the most probable latent values under the model trained by EM.

2.5 Adding Higher-Order Terms

In an RBoW model image regions are treated as being independent. One consequence of this assumption is that RBoW cannot handle fine-grained image regions very well. In this case it is important to use pairwise spatial constraints to ensure coherency and/or capture correlations between labels. In this section we extend RBoW by introducing pairwise dependencies between image regions, making it possible to train models on a fine grid of image regions *e.g.* labeling “at the pixel level”. We propose two variants to RBoW and refer to them as *RBoW+* and *RBoW++*. The former variant uses a Potts model to encourage homogeneous labelings of image regions, whereas the latter variant can capture any arbitrary regular pairwise function.

We denote the pairwise score function by $S_w(z, y)$ and use it to evaluate the goodness of latent configuration z under class y . Note that $S_w(z, y)$ does not depend on the image data x and can be considered as a class conditional prior over latent configurations. This leads to the following score function

$$f_w(x, y) = \max_z \sum_r A_{y,r,z_r} + B_{y,z_r} \cdot \psi(x_r) + S_w(z, y). \quad (2.20)$$

Note that A also does not depend on the image data x and can be absorbed in S_w . However, we chose not to merge them to make the distinction between RBoW and the proposed extensions clear.

The score function of RBoW, RBoW+, and RBoW++ can all be expressed in the general form of (2.20). RBoW does not capture pairwise relationships and we can set $S_w(z, y) = 0$ to get the score function of RBoW (see Equation 2.17). In RBoW+ the pairwise terms are used to encourage coherent labelings and S_w is defined as follows:

$$S_w(z, y) = \sum_r \sum_{s \in \mathcal{N}(r)} T_{z_r} \mathbf{1}\{z_r = z_s\}, \quad (\text{for RBoW+}) \quad (2.21)$$

where $\mathcal{N}(r)$ denotes the set regions that are neighbor to region r , and $T_j \geq 0$ is a parameter of the model that reflects the tendency of the model to keep regions with label j together. We use a regular grid to define image regions and use a 4-connected

neighborhood system to define \mathcal{N} . RBoW++ is more general than RBoW+ and can learn any arbitrary sobmodular pairwise relationship between labels as follows:

$$S_w(z, y) = \sum_r \sum_{s \in \mathcal{N}(r)} C_{y, z_r, z_s}, \quad (\text{for RBoW++}) \quad (2.22)$$

where C_y is a $M \times M$ matrix that parameterizes a regular pairwise relationship (see (2.27) for definition of regularity). In theory, C_y can vary spatially and we can define a different pairwise relationship matrix for each pair of neighboring regions (r, s) . In practice, however, we use two homogeneous pairwise relationship matrices $C^{(H)}$ and $C^{(V)}$ for horizontal and vertical neighbors and define S_w for RBoW++ as

$$S_w(z, y) = \sum_r \sum_{s \in \mathcal{N}^{(H)}(r)} C_{y, z_r, z_s}^{(H)} + \sum_r \sum_{s \in \mathcal{N}^{(V)}(r)} C_{y, z_r, z_s}^{(V)}, \quad (2.23)$$

where $\mathcal{N}^{(H)}$ and $\mathcal{N}^{(V)}$ are the set of horizontal and vertical edges in a 4-connected neighborhood system respectively.

2.5.1 Inference with Graph-Cuts

Inference in the family of RBoW models (including its extended versions) involves finding a labeling $z = (z_1, \dots, z_R)$ that maximizes the score function $f_w(x, y)$ for a given image x and category y . The time complexity of finding the optimal labeling in RBoW is linear in the number of image regions R and in the number of model parts M . However, the task is NP-hard in the case of RBoW+ and RBoW++, and therefore, we resort to approximate solutions.

Inference in RBoW+ and RBoW++ can be seen as MAP inference in a pairwise MRF with respect to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each region r in the image is associated to a node $v_r \in \mathcal{V}$. \mathcal{E} includes the set of edges in a grid graph with 4-connected neighborhood system. In an MRF the energy of a configuration z is defined as follows:

$$E(z) = E_{\text{data}}(z) + E_{\text{prior}}(z). \quad (2.24)$$

MAP inference finds the configuration z that minimizes the energy function E . For pairwise MRFs we have that

$$E_{\text{data}}(z) = \sum_{r \in \mathcal{V}} D_r(z_r) \quad (2.25)$$

$$E_{\text{prior}}(z) = \sum_{(r,s) \in \mathcal{E}} V_{r,s}(z_r, z_s) \quad (2.26)$$

If the pairwise term $V_{r,s}$ satisfies certain properties (see Section 2.5.2 for details) the α/β -swap algorithm of [9] can be used to approximately minimize the energy function. The α/β -swap is an iterative algorithm that runs in polynomial time. In each round, it reduces a multi-label labeling problem to a binary labeling problem and uses graph-cuts to find the global minimum of the reduced binary labeling problem. In each iteration the algorithm chooses a pair of labels (α, β) and updates labels of all nodes that are labeled either α or β in the graph. The update step (*a.k.a.* swap move) can only change a label α to β or vice versa. Any label $\gamma \notin \{\alpha, \beta\}$ remains unchanged during an α/β -swap move. The algorithm stops when there exists no swap move that changes the solution.

For a given category y we define $D_r(z_r) = -A_{y,r,z_r} - B_{y,z_r} \cdot \psi(x_r)$ and $V_{r,s}(z_r, z_s) = -C_{y,z_r,z_s}$ and use the α/β -swap algorithm to do inference in RBoW++. For RBoW+ every thing is the same except that $V_{r,s}(z_r, z_s) = -T_{z_r} \mathbf{1}\{z_r = z_s\}$.

2.5.2 Regular Pairwise Functions

Recall that the α/β -swap algorithm solves an energy minimization problem on binary variables in each iteration. The algorithm can be used to solve energy functions of the form (2.26) only if the pairwise terms $V_{r,s}$ satisfy the following property:

$$V_{r,s}(\alpha, \alpha) + V_{r,s}(\beta, \beta) \leq V_{r,s}(\alpha, \beta) + V_{r,s}(\beta, \alpha), \quad \forall \alpha, \beta \in \{1, \dots, M\}, \quad (2.27)$$

where M denote the size of the label set. In other words, the α/β -swap algorithm can be used only when the pairwise function $V_{r,s}$ yields a *regular* binary energy function

for any choice of (α, β) [32]. For RBoW+ regularity boils down to constraining T_j 's to be non-negative i.e. $T_j \geq 0, \forall j \in \{1, \dots, M\}$. For RBoW++, however, regularity requires enforcing several linear constraints on the model parameters $C_{y,j,k}$. More precisely, (2.27) has to hold for all neighboring regions $(r, s) \in \mathcal{E}$. Note that each constraint in (2.27) involves multiple parameters. While satisfying constraints on combination of multiple parameters may be hard, bounding the values of each parameter individually is relatively easier. In this section we show that we can convert these joint constraints to bounding constraints on single variables. To this end, we write the pairwise function $V_{r,s}$ in the following parametric form:

$$V_{r,s}(j, k) = \beta_{r,s}(j, k) + \alpha_{r,s}(j, k) + \frac{\gamma_{r,s}(j) + \gamma_{r,s}(k)}{2}. \quad (2.28)$$

This particular form parametrization makes it possible to isolate various types of pairwise label-interactions. $\beta_{r,s}$ captures relative geometric relationship between two neighboring regions r and s . It is natural to assume that $\beta_{r,s}(j, k) = \beta_{s,r}(k, j) = -\beta_{r,s}(k, j)$. For example, if region r is on top of region s , the reward for labeling r as *sky* and s as *grass* should be the same as the penalty for switching their labels. This requires that $\beta_{r,s}(j, j) = 0, \forall (r, s) \in \mathcal{E}, \forall j \in \{1, \dots, M\}$. $\alpha_{r,s}$ captures co-occurrence of two labels regardless of the relative geometric configuration of their corresponding regions, thus, $\alpha_{r,s}(j, k) = \alpha_{r,s}(k, j)$. One can represent $\alpha_{r,s}$ as an $M \times M$ matrix. There is a subtle distinction between the diagonal and off-diagonal entries in this matrix. The off-diagonal entries capture *boundary properties* of a region label (part). The diagonal entries, however, capture a measure of integrity of each region label and, thus, can be used to encourage coherent labelings. We set the diagonal entries of α to zero and use $\gamma_{r,s}(a)$ to capture the integrity properties of region labels instead.

To summarize, β and α are $M \times M$ matrices and γ is a vector of length M . β is antisymmetric and α is symmetric with zero diagonal. In practice, we use homogeneous pairwise relations in the horizontal and in the vertical direction parameterized

by $(\alpha^{(H)}, \beta^{(H)}, \gamma^{(H)})$ and $(\alpha^{(V)}, \beta^{(V)}, \gamma^{(V)})$ respectively:

$$V_{r,s}(j, k) = \begin{cases} \beta_{j,k}^{(H)} + \alpha_{j,k}^{(H)} + \frac{\gamma_j^{(H)} + \gamma_k^{(H)}}{2} & \text{if } (r, s) \in \mathcal{E}_H \\ \beta_{j,k}^{(V)} + \alpha_{j,k}^{(V)} + \frac{\gamma_j^{(V)} + \gamma_k^{(V)}}{2} & \text{if } (r, s) \in \mathcal{E}_V \end{cases} \quad (2.29)$$

To enforce regularity we need to substitute $V_{r,s}$ from the previous equation in the regularity condition of Equation 2.27, leading to

$$\beta_{j,j} + \alpha_{j,j} + \frac{2\gamma_j}{2} + \beta_{k,k} + \alpha_{k,k} + \frac{2\gamma_k}{2} \leq \beta_{j,k} + \alpha_{j,k} + \frac{\gamma_j + \gamma_k}{2} + \beta_{k,j} + \alpha_{k,j} + \frac{\gamma_j + \gamma_k}{2} \quad (2.30)$$

$$\iff \alpha_{j,k} \geq 0 \quad (2.31)$$

Thus, to enforce regularity for a pairwise label function of the above form it suffices to constraint α 's to be non-negative. In the next section we show that any regular pairwise function can be decomposed in this form proving that RBoW++ can learn any arbitrary regular function.

2.5.3 Reformulating the Regularity Constraints

In Section 2.5.2 we showed that pairwise functions V that can be decomposed as $V(j, k) = B(j, k) + A(j, k) + 0.5(D(j) + D(k))$ are regular where B is an antisymmetric matrix and A is a non-negative symmetric matrix with zero diagonal. Here we prove that any arbitrary regular function can be written in this form. This proves the equivalence of regularity and this decomposition.

Theorem 2.1. *Any arbitrary regular function V can be written as*

$$V(j, k) = B(j, k) + A(j, k) + \frac{D(j) + D(k)}{2}, \quad (2.32)$$

where B is antisymmetric and A is non-negative and symmetric with zero diagonal.

Proof. Any arbitrary matrix V can be written as the sum of an antisymmetric matrix

B and a symmetric matrix A' as follows:

$$V = \frac{V - V^T}{2} + \frac{V + V^T}{2} \quad (2.33)$$

$$= B + A'. \quad (2.34)$$

Let $D = \text{diag}(A')$ be the diagonal of matrix A' . We can write A' as

$$A'(j, k) = \frac{D(j) + D(k)}{2} + A(j, k) \quad (2.35)$$

Note that A is symmetric (since A' was symmetric) and $A(j, j) = A'(j, j) - \frac{2D(j)}{2} = 0$.

In summary:

$$B = \frac{V - V^T}{2} \quad \text{antisymmetric} \quad (2.36)$$

$$D = \text{diag}\left(\frac{V + V^T}{2}\right) \quad (2.37)$$

$$A = \frac{V + V^T}{2} - \frac{D\mathbf{1}_M^T + \mathbf{1}_M D^T}{2} \quad \text{symmetric with zero diagonal} \quad (2.38)$$

where $\mathbf{1}_M$ is the column vector of all ones of length M . □

2.6 Experimental Results on Scene Classification

We evaluate the RBoW model on the *15 Scene* dataset from [35] and on the *MIT 67 Indoor Scenes* dataset from [52]. We measured the performance of different models using the average of the diagonal entries of their confusion matrix.

We extract densely sampled SIFT features [41] as local descriptors, and create a visual vocabulary using *k-means* clustering on a subset of the SIFT features randomly sampled from training images. We set the size of the visual vocabulary to be $K = 200$ in all of our experiments.

For discriminative training we used a feature map $\psi(b)$ that normalizes the bag of words vector b to have unit norm and then computes the square root of each entry.

All of the experiments with SBoW and RBoW models used a 4×4 regular grid to

15 Scenes	BoW	SBoW	RBoW		
			Init-rand	Init-fixed	Init-EM
Disc.	71.7 ± 0.2	77.7 ± 0.9	74.5 ± 0.4	78.5 ± 1.1	78.6 ± 0.7
Gen.	62.1 ± 2.4	74.3 ± 0.5	76.1 ± 0.5		

Table 2.1: Average performance of different methods on the 15 scene dataset. We used three different initialization methods for training a discriminative RBoW model.

partition the image into $R = 16$ rectangular regions. For the RBoW models we used $M = 16$ region models for each image category. Taking $M = R$ makes it possible to initialize an RBoW model with a fixed assignment of region models to image regions, with one region model for each image region.

2.6.1 15 Scene Dataset

The 15 Scene dataset contains 4485 images of 15 different scenes. It includes both indoor scenes (*office, bedroom, kitchen, living-room, store*) and outdoor scenes (*sub-urb, coast, forest, highway, inside-city, mountain, open-country, street, tall-building, industrial*). The dataset does not provide separate training and test sets, so we use 5 random splits and compute the mean and standard deviation of the classification performance across splits. In each split we use 100 training images for each category.

Table 2.1 compares the overall performance of RBoW to the SBoW and BoW baselines. Again we see that careful initialization is important for LS-SVM training. Initialization of CCCP using a generative model trained with EM leads to the best performance, while random initialization leads to the worst performance.

2.6.2 MIT 67 Indoor Scenes

The MIT dataset contains images from 67 different categories of indoor scenes. There is a fixed training and test set containing approximately 80 and 20 images from each category respectively.

Table 2.2 summarizes the performance of our models and some previously published methods. In [45] classification scores from a deformable part model (DPM) [20] for each category are combined with that of color GIST descriptors [44], together

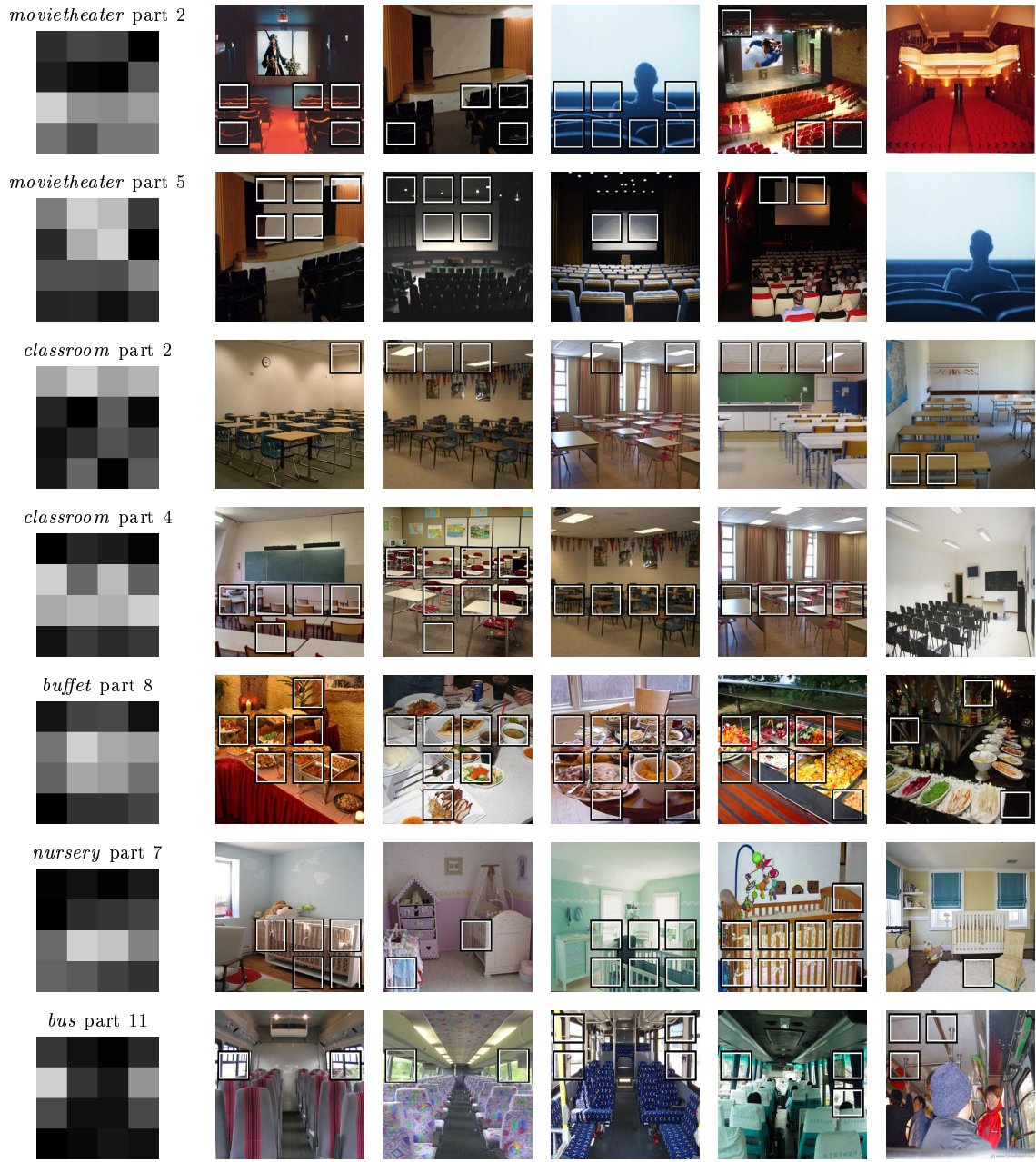


Figure 2-3: Some interesting region models learned for different categories using a discriminative RBoW model (Init-EM). Each row illustrates a region model for a particular category. The first column shows the preferences of different image regions for this region model ($A_{y,r,j}$ for fixed y and j). The other columns show image regions that were assigned to this region model during classification ($z_r = j$).

MIT 67 Indoor Scenes	Method	Rate	LS-SVM Objec.
Prev. Works	ROI+GIST [52]	26.5	
	MM-scene [74]	28.0	
	CENTRIST [68]	36.9	
	Object Bank [37]	37.6	
	DPM [45]	30.4	
	DPM+GIST-color+SP [45]	43.1	
BoW	Generative	12.80	
	Discriminative	25.17	
SBoW	Generative	19.46	
	Discriminative	33.99	
RBoW	Generative	27.66	
	Discriminative Init-rand	31.63	91.08
	Discriminative Init-fixed	34.99	83.50
	Discriminative Init-EM	37.93	80.30

Table 2.2: Average performance of different methods on the MIT dataset. The last column shows the final value of the LS-SVM objective function for RBoW models with different initializations.

with spatial pyramid matching [35]. To compare reconfigurable models to deformable models we also include the performance obtained in [45] using DPMs alone. Table 2.2 also shows the performance of our BoW and SBoW baselines. The performance gap between the BoW and SBoW approaches proves a considerable point regarding the importance of spatial information for image classification.

Table 2.2 includes the results of discriminative RBoW models trained with different initialization methods. As discussed in Section 2.4.2, CCCP requires initial latent values for each training example. *Init-rand* selects random initial region models for each image region. *Init-fixed* selects a fixed initial region model for each image region. *Init-EM* uses a generative RBoW model trained with EM, and selects the most probable latent values under the generative model to initialize LS-SVM training. We have found that Init-EM gives consistently better results. This shows the importance of initialization for LS-SVM training. It also shows that while generative models typically don’t perform as well as discriminative models, EM training seems to be less susceptible to local optima when compared to LS-SVM.

Last column of Table 2.2 shows the final value of LS-SVM objective under each

initialization method. Note that the value of the objective is consistent with the performance of the model, suggesting that developing better optimization algorithms for LS-SVM should lead to better models.

Table 2.3 shows per-category performance of the discriminative RBoW model (initialized with EM), the discriminative baseline approaches and the DPM method from [45]. Note that although SBoW has a better overall accuracy than BoW, it does worse in 12 classes. RBoW is able to recover the performance lost by SBoW in several classes, including *florist*, *gameroom* and *videostore*. RBoW model performs significantly better than our baselines and the DPM method of [45] on several classes.

Figure 2-3 illustrates some interesting region models that were learned for different categories. For example, in the *buffet* class there is a model for food regions, in the *nursery* class there is a model for crib regions, while in the *classroom* class there is a model for regions with desks and another for the ceiling.

Training RBoW models is reasonably fast. Training a generative RBoW model with EM (with 16 parts and 16 image regions) on the MIT dataset takes about 10 minutes on a 2.8GHz computer with an i7 multi-core processor. Training a similar discriminative model with LS-SVM on the MIT dataset takes about 10 hours. Discriminative training takes much longer than EM because step 2 of CCCP involves a large convex optimization problem. At test time our implementation can classify more than 180 images per second for the MIT dataset. The running time for classification scales linearly with the number of classes.

2.6.3 Synthetic Scene Dataset

Although RBoW can only capture absolute spatial properties, such as *sky* tends to appear on the top of image in *beach* scenes, RBoW++ models can capture relative geometric relationships between parts (*e.g.* *sky* tends to appear on top of *sea* in *beach* scene). It can also learn relations between composite objects [57] such as *person* on top of *horse*. In order to demonstrate the strength of our model and showcase the types of properties it can capture we test it on a synthetic dataset. We compare performance of RBoW++, RBoW+, RBoW, SBoW, and BoW models on this synthetic dataset.

Category	RBoW	SBoW	BoW	DPM	Category	RBoW	SBoW	BoW	DPM
bowling	85	85	55	35	dentaloffice	48	29	19	24
florist	84	63	74	79	casino	47	47	63	32
ins. subway	81	62	57	62	gameroom	45	10	40	40
cloister	80	85	55	90	prisoncell	45	40	35	40
inside bus	78	61	9	43	trainstation	45	70	35	35
greenhouse	75	80	80	65	auditorium	44	39	22	11
church ins.	74	79	53	63	bar	44	39	33	11
classroom	72	56	44	67	clothingstore	44	33	11	33
buffet	65	60	55	75	garage	44	39	39	56
concert hall	65	60	55	65	corridor	43	62	33	57
elevator	62	62	57	52	meetingroom	41	55	27	75
closet	61	56	56	44	videostore	41	18	23	18
comp. room	56	33	6	22	hospitalroom	40	30	5	5
movietheater	55	65	50	45	kindergarden	40	40	25	15
nursery	55	50	75	60	museum	39	26	0	13
pantry	55	50	30	75	kitchen	38	43	14	29
library	50	45	40	0	studiomusic	37	37	11	32
Category	RBoW	SBoW	BoW	DPM	Category	RBoW	SBoW	BoW	DPM
laundromat	36	41	9	45	airport ins.	20	15	15	5
stairscase	35	35	45	35	bedroom	19	14	0	5
bathroom	33	22	6	50	hairsalon	19	24	5	43
grocerystore	33	29	38	19	locker room	19	14	14	19
subway	33	33	14	38	warehouse	19	19	10	24
bookstore	30	20	30	45	artstudio	15	5	0	5
winecellar	29	29	29	14	toystore	14	5	0	9
child. room	28	28	11	6	lobby	10	10	5	30
dining room	28	22	11	28	poolinside	10	5	5	0
gym	28	6	0	22	restaurant	10	10	10	5
lab. wet	27	18	0	5	office	10	10	10	10
rstrnt kitchen	26	26	0	4	bakery	5	5	0	11
mall	25	15	10	25	operat. room	5	0	32	5
waitingroom	24	14	5	5	livingroom	5	5	10	20
fastfoodrstrnt	24	6	35	12	deli	0	0	5	5
tv studio	22	44	17	6	jewel. shop	0	9	0	5
shoeshop	21	32	21	16					

Table 2.3: Performance of our reconfigurable model and different baseline methods on the MIT dataset. The last column shows performance of DPM method from [45].

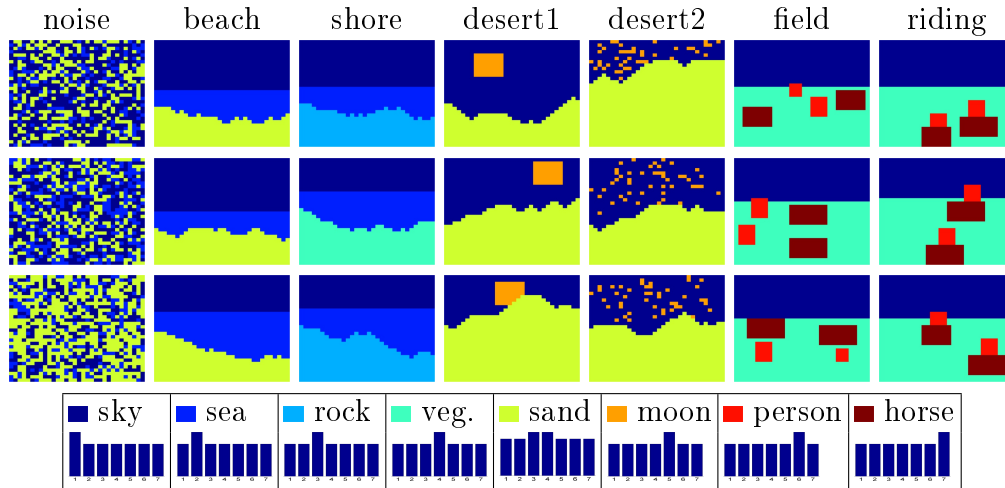


Figure 2-4: Sample images from the synthetic dataset. Each column shows three images from one scene category. The semantic meaning of each label together with the corresponding observation distribution is shown at the bottom.

This comparison helps us understand the differences between these models and the types of properties they can capture and the cases where they fail.

The synthetic dataset consists of 7 hypothetical scene categories: *noise*, *beach*, *shore*, *desert1*, *desert2*, *field*, and *(horseback-)riding*. Each image in the synthetic dataset is made of a grid of 32×32 regions. Each region is labeled by one of 8 parts: *sky*, *sea*, *rock*, *vegetation*, *sand*, *star/moon*, *person*, and *horse*. An image is represented as an ordered set of 32×32 feature vectors (one for each image region). The feature vector associated to a region is a bag of words histogram computed from 1000 discrete observations. The observations within a region are generated from a multinomial distribution defined by the type of the region (identified by the label given to it). We define the multinomial distributions associated to each one of the 8 parts (except for *sand*) to be slightly peaked at a unique observation and uniform otherwise (see Figure 2-4-bottom). The observation distribution for *sand* is the average of the observation distributions for *vegetation* and *rock*. The reason for this choice is to demonstrate strength of RBoW models (and their extended versions) over BoW and SBoW in learning mixtures of parts (see Figure 2-5 for a quantitative comparison).

There are 1400 training and 1400 test images (200 per category) in the dataset. Images are generated by randomly labeling each region in a 32×32 grid according

to the category of the image. Each category has a particular labeling process which defines the signature layout of the category. For example, the labeling process of the *beach* category starts by setting the ocean’s horizon as a horizontal line located, uniformly at random, along rows 8-18 of the grid. Then a curved shoreline is chosen at random below the horizon. Finally, all regions on and above the horizon are labeled as *sky*, regions below horizon and above (or on) the shoreline are labeled as *sea*, and the regions below the shoreline are labeled as *sand* (see exemplar images in the second column in Figure 2-4). Given a labeling for all image regions we generate local observations within each region by drawing random samples from the multinomial distribution corresponding to the part assigned to the image region. Finally, we compute a feature vector (BoW histogram) for each region in the image.

In the following we briefly explain the labeling process of each of the 7 categories that constitutes the layout of the scene categories in the dataset. Three sample images from each category are shown in Figure 2-4.

Recall that each image is made of a grid of 32×32 regions. We explained signature layout of the *beach* category before. In the *noise* category image regions are labeled randomly from the set of three labels *sky*, *sea*, *sand*. We make sure that the average frequency of each part in the images of *noise* category is the same as that of the *beach* scene. Due to the absence of spatial information in the BoW model we expect it to fail in discriminating between *noise* and *beach*.

shore is similar to *beach* in all aspects except that there are no *sand* regions in *shore*; *sand* regions are replaced with *rock* in some images and *vegetation* in some others. In other words, the land in the *shore* scene is a mixture of *rock* and *vegetation*. We expect SBoW to fail in discriminating between *beach* and *shore*. In order to intensify the confusion between the two categories we set the observation distribution associated to *sand* to be the average of the observation distributions of *rock* and *vegetation* (see Figure 2-4-bottom).

In *desert1* a curved line (indicating sandhills) splits the image into two areas; *sky* on the top and *sand* on the bottom. There is a fixed size rectangle, representing the *moon* in a random location in the sky in all images of this category. The *desert2*

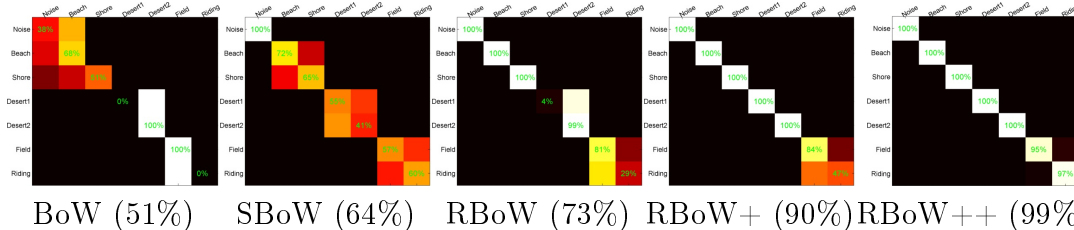


Figure 2-5: Confusion matrices corresponding to different models. The average of the diagonal terms is shown in the parentheses.

category is similar to *desert1* in all aspects except that the *moon* is replaced with little *stars* scattered randomly in the sky. We have that the *moon* and *star* both share the same observation distribution. We expect RBoW to fail in discriminating between *desert1* and *desert2* because it assumes that regions are independent, and thus, cannot tell a coherent labeling apart from a non-coherent one.

In the *field* category a horizon line (located at row 6 to 14 of the grid uniformly at random) separates *sky* from the ground which is labeled as *vegetation*. Two *horses* and two *persons* with various sizes are placed at random locations on the ground. The *riding* category is the same as the *field* category except that the *persons* are on top of the *horses* in the *riding* category. RBoW+ fails to discriminate between these two categories because it cannot capture relative geometric relations between parts.

We compare performance of BoW, SBoW, RBoW, RBoW+, and RBoW++ models on the synthetic dataset in Figure 2-5. BoW classifies images into three subsets namely $\{noise, beach, shore\}$, $\{desert1, desert2\}$, $\{field, riding\}$, and cannot disambiguate the categories within each set. In SBoW, *beach* is confused with *shore*, *desert1* is confused with *desert2*, and *field* is confused with *riding*. RBoW does better than SBoW by successfully telling apart *beach* and *shore*. RBoW+ only confuses *field* and *riding* categories. Finally, RBoW++ distinguishes all categories almost perfectly.

Recall from Section 2.4.2 that the training objective of RBoW and its extensions (*i.e.* RBoW+ and RBoW++) are non-convex, and since we use Algorithm 2 from Chapter 1 for training, the quality of the trained models depends on how the training is initialized. In all our experiments we initialized the model by setting $\mathbf{z}^{(0)}$ so that all image regions get the ground-truth part assigned to them.

2.7 Conclusion

Reconfigurable models represent images by a collection of regions with specific content. For each scene category we have a set of region models. The content of an image is defined by latent variables that assign a region model to each image region. We introduce different versions of the reconfigurable model. The most basic version, called RBoW, assumes the latent variables are independent conditional on the image category. The most powerful version, called RBoW++, breaks this assumption and can capture any regular pairwise relationship between region labels (parts). For example, RBoW++ can avoid regions on top of *sky* to be labeled as *water*.

Our current models rely on a pre-defined partition of an image into a grid of regions. We would like to relax this assumption so that we can better capture the content of an image.

Latent variable models lead to challenging training problems, especially in the discriminative setting. Our experiments demonstrate that EM can be used as an effective method for initializing LS-SVM training.

Chapter 3

Automatic Part Discovery from Image Collections

Part-based representations have been shown to be very useful for image classification. Learning part-based models is often viewed as a two-stage problem. First, a collection of informative parts is discovered, using heuristics that promote part distinctiveness and diversity, and then classifiers are trained on the vector of part responses. We unify the two stages and learn the image classifiers and a set of shared parts jointly. Initially we generate a pool of parts by randomly sampling part candidates and selecting a good subset using ℓ_1/ℓ_2 regularization. All steps are driven *directly* by the same objective namely the classification loss on a training set. This lets us do away with engineered heuristics. We also introduce the notion of *negative parts*, intended as parts that are negatively correlated with one or more classes. Negative parts are complementary to the parts discovered by other methods, which look only for positive correlations.

3.1 Introduction

Computer vision makes abundant use of the concept of “part”. There are at least three key reasons why parts are useful for representing objects or scenes. One reason is the existence of non-linear and non-invertible nuisance factors in the generation of images, including occlusions. By breaking an object or image into parts, unoccluded regions

become recognizable. A second reason is that parts can be recombined in a model to express a combinatorial number of variants of an object or scene. For example parts corresponding to objects (*e.g.* a laundromat and a desk) can be rearranged in a scene, and parts of objects (*e.g.* the face and the clothes of a person) can be replaced by other parts. A third reason is that parts are often distinctive of a particular (sub)category of objects (*e.g.* cat faces usually belong to cats).

Discovering good parts is a difficult problem that has recently raised considerable interest [31, 17, 63]. The quality of a part can be defined in different ways. Methods such as [31, 17] decouple learning parts and image classifiers by optimizing an intermediate objective that is only heuristically related to classification. Our first contribution is to learn a *system of discriminative parts jointly with the image classifiers*, optimizing the overall classification performance on a training set. We propose a unified framework to train all the model parameters jointly (Section 3.5), and show that joint training can substantially improve the quality of the models (Section 3.8).

A fundamental challenge in part learning is a chicken-and-egg problem: without an appearance model, examples of a part cannot be found, and without having examples an appearance model cannot be learned. To address this methods such as [31, 18] start from a single random example to initialize a part model, and alternate between finding more examples and retraining the part model. As the quality of the learned part depends on the initial random seed, thousands of parts are generated and a distinctive and diverse subset is extracted by means of some heuristic. Our second contribution is to propose a simple and effective alternative (Section 3.6). We still initialize a large pool of parts from random examples; we use these initial part models,

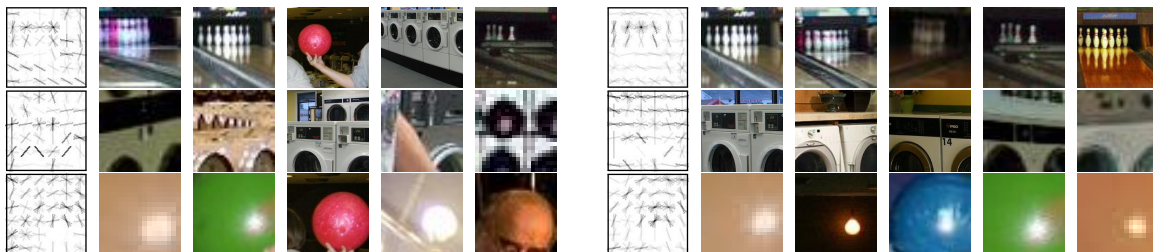


Figure 3-1: Part filters before (left) and after joint training (right) and top scoring detections for each.

Random Part Initialization	Part Selection	Joint Training
<ol style="list-style-type: none"> 1. Extract feature from a patch at random image and location. 2. Whiten the feature. 3. Repeat to construct a pool of candidate parts. 	<ol style="list-style-type: none"> 1. Train part weights u with ℓ_1/ℓ_2 regularization. 2. Discard parts that are not used according to u. 	<ol style="list-style-type: none"> 1. Train part weights u keeping part filters w fixed. 2. Train part filters w keeping part weights u fixed. 3. Repeat until convergence.

Figure 3-2: Our pipeline. Part selection and joint training are driven by classification loss. Part selection is important because joint training is computationally demanding.

each trained from a single example, to train image classifiers using ℓ_1/ℓ_2 regularization as in [63]. This removes uninformative and redundant parts through group sparsity. This simple method produces better parts than more elaborate alternatives. Joint training (Section 3.8) improve the quality of the parts further.

Our pipeline, comprising random part initialization, part selection, and joint training is summarized in Figure 3-2. In Section 3.8 we show empirically that, although our part detectors have the same form as the models in [31, 63], they can reach a higher level of performance using a fraction of the number of parts. This translates directly to test time speedup. We also demonstrate importance of flip-invariant representation for image classification obtained by averaging features extracted from an image and its right-to-left flip as done in [17]. We present experiments with both HOG [14] and CNN [33] features and improve the state-of-the-art results on the MIT-indoor dataset [52] using CNN features.

A final contribution of our work is the introduction of the concept of negative parts, *i.e.* parts that are negatively correlated with respect to a class (Section 3.2). These parts are still informative as “counter-evidence” for the class. In certain formulations, negative parts are associated to negative weights in the model and in others with negative weight differences.

Related Work

Related ideas in part learning have been recently explored in [60, 31, 63, 17]. The general pipeline in all of these approaches is a two-stage procedure that involves pre-

training a set of discriminative parts followed by training a classifier on top of the vector of the part responses. The differences in these methods lay in the details of how parts are discovered. Each approach uses a different heuristic to find a collection of parts such that each part scores high on a subset of categories (and therefore is discriminative) and, collectively, they cover a large area of an image after max-pooling (and therefore are descriptive). Our goal is similar, but we achieve part diversity, distinctiveness, and coverage as natural byproducts of optimizing the “correct” objective function, *i.e.* the final image classification performance.

Reconfigurable Bag of Words (RBoW) model [47] is another part-based model used for image classification. RBoW uses latent variables to assign part models to image regions. In contrast, the latent variables in our model assign image regions to parts via max-pooling.

It has been shown before [25] that joint training is important for the success of part-based models in object detection. Differently from them, however, we share parts among multiple classes and define a joint optimization in which multiple classifiers are learned concurrently. In particular, the same part can vote strongly for a subset of the classes and against another subset.

The most closely related work to ours is [40]. Their model has two sets of parameters; a dictionary of visual words θ and a set of weights u that specifies the importance the visual words in each category. Similar to what we do here, [40] trains u and θ jointly (visual words would be the equivalent of part filters in our terminology). However, they assume that u is non-negative. This assumption does not allow for “negative parts” as we describe in Section 3.2. Our method and [40]’s both require recomputing maximum part responses on training images multiple times during training which is computationally very expensive. We use a caching mechanism to considerably reduce the amount of this computation. We also prove that the caching mechanism leads to an exact solution.

The concept of negative parts and relative attributes [46] are related in that both quantify the relative strength of visual patterns. Our parts are trained jointly using image category labels as the only form of supervision, whereas the relative

attributes in [46] were trained independently using labeled information about the strength of hand picked attributes in training images. More importantly, we train negative parts jointly with image classifiers whereas the relative attributes in [46] are pre-trained.

3.2 Part-Based Models for Image Classification

We model an image class using a collection of parts. A part may capture the appearance of an entire object (*e.g.* bed in a *bedroom* scene), a part of an object (*e.g.* drum in the *laundromat* scene), a rigid composition of multiple objects (*e.g.* rack of clothes in a *closet* scene), or a region type (*e.g.* ocean in a *beach* scene).

Let x be an image. We use $H(x)$ to denote the space of latent values for a part. In our experiments $H(x)$ is a set of positions and scales in a scale pyramid. To test if the image x contains part j at location $z_j \in H(x)$, we extract features $\psi(x, z_j)$ and take the dot product of this feature vector with a part filter w_j . Let $s(x, z_j, w_j)$ denote the response of part j at location z_j in x . Since the location of a part is unknown, it is treated as a latent variable which is maximized over. This defines the response $r(x, w_j)$ of a part in an image,

$$s(x, z_j, w_j) = w_j \cdot \psi(x, z_j), \quad r(x, w_j) = \max_{z_j \in H(x)} s(x, z_j, w_j). \quad (3.1)$$

Given a collection of m parts $w = (w_1, \dots, w_m)$, their responses are collected in an m -dimensional vector of part responses $r(x, w) = (r(x, w_1); \dots; r(x, w_m))$. In practice, filter responses are pooled within several distinct *spatial subdivisions* [35] to encode weak geometry. In this case we have R pooling regions and $r(x, w)$ is an mR -dimensional vector maximizing part responses in each pooling region. For the rest of this chapter we assume a single pooling region to simplify notation.

Part responses can be used to predict the class of an image. For example, high response for “bed” and “lamp” would suggest the image is of a “bedroom” scene.

3.3 Negative Parts in Binary Classifiers

In binary classification the goal is to predict if an image belongs to a foreground class $y = +1$ (“bedroom”) or a background class $y = -1$ (“everything else”). Binary classifiers are often used for multi-class classification with a one-vs-all setup. DPMs [20] also use binary classifiers to detect objects of each class. For a binary classifier we can define a score function $f_\beta(x)$ for the foreground hypothesis. The score combines part responses using a vector of part weights u ,

$$f_\beta(x) = u \cdot r(x, w), \quad \beta = (u, w). \quad (3.2)$$

The binary classifier predicts $y = +1$ if $f_\beta(x) \geq 0$, and $y = -1$ otherwise.

Definition 3.1. *If $u_j > 0$ we say part j is a positive part for the foreground class and if $u_j < 0$ we say part j is a negative part for the foreground class.*

Intuitively, a negative part provides counter-evidence for the foreground class; *i.e.* $r(x, w_j)$ is negatively correlated with $f_\beta(x)$. For example, since cows are not usually in offices a high response from a *cow* filter should penalize the score of a *office* classifier.

Let $\beta = (u, w)$ be the parameters of a binary classifier. We can multiply w_j and divide u_j by a positive value α to obtain an equivalent model. If we use $\alpha = |u_j|$ we obtain a model where $u \in \{-1, +1\}^m$. However, in general it is not possible to transform a model where u_j is negative into a model where u_j is positive because of the max in (3.1). Moreover, allowing real valued u matrix has the advantage that it simplifies part sharing when we turn to multi-class classifiers. It allows two classes to share a part filter with different levels of importance (or weight).

We note that, when $u_j \geq 0, \forall j$ the score function $f_\beta(x)$ is convex in w , and (3.2) reduces to a latent SVM score function, and a special case of a DPM [20]. Otherwise, *i.e.* if there are negative parts, $f_\beta(x)$ is no longer convex in w . In particular, when u is non-negative we can assume $u = \mathbf{1}$ and (3.2) reduces to

$$f_\beta(x) = \sum_{j=1}^m \max_{z_j \in H(x)} w_j \cdot \psi(x, z_j) = \max_{z \in Z(x)} w \cdot \Psi(x, z), \quad (3.3)$$

where $Z(x) = H(x)^m$, and $\Psi(x, z) = (\psi(x, z_1); \dots; \psi(x, z_m))$. In the case of a DPM, the feature vector $\Psi(x, z)$ and the model parameters contain additional terms capturing spatial relationships between parts. In a DPM all part responses are positively correlated with the score of a detection. Therefore DPMs do not use negative parts.

3.4 Negative Parts in Multi-Class Classifiers

In the previous section we showed that certain one-vs-all part-based classifiers, including DPMs, cannot capture counter-evidence from negative parts. This limitation can be addressed by using more general models with two sets of parameters $\beta = (u, w)$ and a score function $f_\beta(x) = u \cdot r(x, w)$, as long as we allow u to have negative entries.

Now we consider the case of a multi-class classifier where part responses are weighted differently for each category but all categories share the same set of part filters. A natural consequence of part sharing is that a positive part for one class can be used as a negative part for another class.

Let $\mathcal{Y} = \{1, \dots, n\}$ be a set of n categories. A multi-class part-based model $\beta = (w, u)$ is defined by m part filters $w = (w_1, \dots, w_m)$ and n vectors of part weights $u = (u_1, \dots, u_n)$ with $u_y \in \mathbb{R}^m$. The shared filters w and the weight vector u_y define parameters $\beta_y = (w, u_y)$ for a score function for class y . For an input x the multi-class classifier selects the class with highest score

$$\hat{y}_\beta(x) = \arg \max_{y \in \mathcal{Y}} f_{\beta_y}(x) = \arg \max_{y \in \mathcal{Y}} u_y \cdot r(x, w) \quad (3.4)$$

We can see u as an $n \times m$ matrix. Adding a constant to a column of u does not change the differences between scores of two classes $f_{\beta_a}(x) - f_{\beta_b}(x)$. This implies the function \hat{y} is invariant to such transformations. We can use a series of such transformations to make all entries in u non-negative, without changing the classifier. Thus, in a multi-class part-based model, unlike the binary case, it is not a significant restriction to require the entries in u to be non-negative. In particular the sign of an entry in u_y does not determine the type of a part (positive or negative) for class y .

Definition 3.2. If $u_{a,j} > u_{b,j}$ we say part j is a positive part for class a relative to b . If $u_{a,j} < u_{b,j}$ we say part j is a negative part for class a relative to b .

Although adding a constant to a column of u does not affect \hat{y} , it does impact the norms of the part weight vectors u_y . For an ℓ_2 regularized model the columns of u will sum to zero. Otherwise we can subtract the column sums from each column of u to decrease the ℓ_2 regularization cost without changing \hat{y} and therefore the classification loss. We see that in the multi-class part-based model constraining u to have non-negative entries only affects the regularization of the model.

3.5 Joint Training

In this section we propose an approach for joint training of all parameters $\beta = (w, u)$ of a multi-class part-based model. Training is driven directly by classification loss. Note that a classification loss objective is sufficient to encourage diversity of parts. In particular joint training encourages part filters to complement each other. We have found that joint training leads to a substantial improvement in performance (see Section 3.8). The use of classification loss to train all model parameters also leads to a simple framework that does not rely on multiple heuristics.

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^k$ denote a training set of labeled examples. We train β using ℓ_2 regularization for both the part filters w and the part weights u (we think of each as a single vector) and the multi-class hinge loss, resulting in the objective function:

$$O(u, w) = \lambda_w \|w\|^2 + \lambda_u \|u\|^2 + \sum_{i=1}^k \max \left\{ 0, 1 + (\max_{y \neq y_i} u_y \cdot r(x_i, w)) - u_{y_i} \cdot r(x_i, w) \right\} \quad (3.5)$$

$$= \lambda_w \|w\|^2 + \lambda_u \|u\|^2 + \sum_{i=1}^k \max \left\{ 0, 1 + \max_{y \neq y_i} (u_y - u_{y_i}) \cdot r(x_i, w) \right\} \quad (3.6)$$

We use block coordinate descent for training, as summarized in Algorithm 3. This alternates between (Step 1) optimizing u while w is fixed, and (Step 2) optimizing w while u is fixed. The first step reduces to a convex structural SVM problem (line 3 of Algorithm 3). If u is non-negative the second step also reduces to a latent structural

Algorithm 3 Joint training of model parameters by optimizing $O(u, w)$ in Equation 3.6.

Input: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$

1: initialize the part filters $w = (w_1, \dots, w_m)$

2: **repeat**

3: $u := \arg \min_{u'} O(u', w)$ (defined in Equation 3.6)

4: **repeat**

5: $w^{\text{old}} := w$

6: $w := \arg \min_{w'} B_u(w', w^{\text{old}})$ (defined in Equation 3.8)

7: **until** convergence

8: **until** convergence

Output: $\beta = (u, w)$

SVM problem defined by (3.5). We use a novel approach that allows u to be negative and uses an equivalent interpretation of the objective function defined by (3.6).

Step 1: Learning Part Weights (line 3 of Algorithm 3)

This involves computing $\arg \min_{u'} O(u', w)$. Since w is fixed $\lambda_w \|w\|^2$ and $r(x_i, w)$ are constant and, therefore, can be ignored in optimization of classifiers. This makes the learning of the part weights equivalent to training a multi-class SVM where the i -th training example is represented by an m -dimensional vector of part responses $r(x_i, w)$. This is a convex problem that can be solved efficiently using standard methods, including the ones discussed in Section 1.5.

Step 2: Learning Part Filters (lines 4-7 of Algorithm 3)

This involves solving the following optimization problem while u is kept fixed:

$$\begin{aligned}
 & \arg \min_w O(u, w) \\
 &= \arg \min_w \lambda_w \|w\|^2 + \lambda_u \|u\|^2 + \sum_{i=1}^k \max \left\{ 0, 1 + \max_{y \neq y_i} (u_y - u_{y_i}) \cdot r(x_i, w) \right\} \\
 &= \arg \min_w \lambda_w \|w\|^2 + \sum_{i=1}^k \max \left\{ 0, 1 + \max_{y \neq y_i} (u_y - u_{y_i}) \cdot r(x_i, w) \right\}. \tag{3.7}
 \end{aligned}$$

The last step uses the fact that u is kept fixed during the optimization of w .

While $r(x_i, w_j)$ is convex in w (it is a maximum of linear functions) the coefficients $u_{y,j} - u_{y_i,j}$ may be negative, making the objective function of (3.7) non-convex. We solve (3.7) via sequential convex bound optimization; which is implemented in lines 4-7 of Algorithm 3. In each iteration we construct a convex upper bound to the objective function of (3.7) using the current estimate of w and update w to minimize the bound. The key to convergence of this iterative procedure is that the value of the objective function is guaranteed to decrease after each iteration.

Let w^{old} denote the current estimate of the part filters. We denote the bound by $B_u(w, w^{\text{old}})$ and introduce the following notation to define it. Let $s(x_i, z_i, w) = (s(x_i, z_{i,1}, w_1); \dots; s(x_i, z_{i,m}, w_m))$ be the vector of part responses in image x when the latent variables are fixed to $z_i = (z_{i,1}, \dots, z_{i,m})$. We replace $r(x_i, w_j)$ with $s(x_i, z_{i,j}, w_j)$ in (3.7) when $u_{y,j} - u_{y_i,j} < 0$ and make the bound tight at w^{old} by selecting $z_{i,j} = \arg \max_{z_j \in H(x_i)} s(x_i, z_j, w_j^{\text{old}})$. Finally, we define the bound as follows:

$$B_u(w, w^{\text{old}}) = \lambda_w \|w\|^2 + \sum_{i=1}^k \max \left\{ 0, 1 + \max_{y \neq y_i} (u_y - u_{y_i}) \cdot [S_{y,y_i} r(x_i, w) + \bar{S}_{y,y_i} s(x_i, z_i, w)] \right\}, \quad (3.8)$$

where, for a pair of categories (y, y') , $S_{y,y'}$ and $\bar{S}_{y,y'}$ are $m \times m$ diagonal 0-1 matrices that select $r(x_i, w_j)$ when $u_{y,j} - u_{y_i,j} \geq 0$ and $s(x_i, z_{i,j}, w_j)$ when $u_{y,j} - u_{y_i,j} < 0$. Specifically, $S_{y,y'}(j, j) = 1$ if and only if $u_{y,j} - u_{y',j} \geq 0$ and $\bar{S}_{y,y'}(j, j) = 1 - S_{y,y'}(j, j)$.

One can verify that $B_u(w, w^{\text{old}})$ is convex, upper bounds the objective function of (3.7), and touches the objective function of (3.7) at $w = w^{\text{old}}$.

Optimization of the bound B_u (which is done in line 6 of Algorithm 3) requires significant computational and memory resources. In the next section we describe how we solve this optimization problem in practice.

3.5.1 Caching Hard Examples

Optimizing the bound function $B_u(w, w^{\text{old}})$ is computationally expensive because of two reasons. Firstly, the number of shared parts filters is large. Secondly, it requires

convolving the part filters with the training images repeatedly which is a slow process. We speed up the optimization of B_u by bypassing the convolution process. More specifically, we use a caching mechanism that makes it sufficient to compute the score of the part filters on a very small subset of locations in the scale pyramid of the training images. This is very similar to the caching technique used in training DPMS [20]. Note that here, unlike [20], there is no notion of hard *negatives* because we use a multi-class classification objective. Instead we have hard *examples*. A hard example is a training example along with the best assignment of its latent variables (with respect to a given model) that either has non-zero loss or lies on the decision boundary. In the following we explain our caching mechanism and prove it converges to the unique global minimum of the bound.

The caching mechanism is iterative where we alternative between 1) finding the optimal part filters subject to the data in the cache, and 2) updating the content of the cache. This procedure is guaranteed to converge to the *global* minimum of $B_u(w, w^{old})$. Initially, we start from an empty cache. To update the cache, we remove all the “easy” entries from the cache and add new “hard” entries to it.

In each iteration, the cache stores a small number of *active* locations from the scale hierarchy of each image for each part. Thus, finding the highest responding location of each part among the active entries in the cache requires only a modest amount of computation. Although the caching mechanism requires multiple rounds of optimization, however, each optimization problems is rather simple and, overall, the problem of optimizing B_u becomes tractable.

Let $Z_i = \{(y, z) : y \in \mathcal{Y}, z \in H(x_i)^m\}$ be the set of all possible latent configurations of m parts on image x_i . Also, let $z, \bar{z} \in H(x)^m$ denote two arbitrary placements of *all* the parts. Also, let $a, \bar{a} \in \mathbb{R}^m$ denote two part weight vectors such that $a_j \bar{a}_j = 0, \forall j \in \{1, \dots, m\}$; that is, if $a_j \neq 0$ then $\bar{a}_j = 0$, and vice versa. We use $\Phi(x, z, \bar{z}, a, \bar{a})$ to denote the feature function that chooses a placement for part j between z_j and \bar{z}_j and concatenates features extracted from placing all of the m parts on image x . The feature function outputs an md -dimensional feature vector and is

defined as follows

$$\Phi(x, z, \bar{z}, a, \bar{a}) = (a_1\psi(x, z_1) + \bar{a}_1\psi(x, \bar{z}_1); \dots; a_m\psi(x, z_m) + \bar{a}_m\psi(x, \bar{z}_m)).$$

We use the notation used in the previous section and define two part weight vectors a_{y,y_i} and \bar{a}_{y,y_i} as follows

$$a_{y,y_i} = (u_y - u_{y_i})^T S_{y,y_i}$$

$$\bar{a}_{y,y_i} = (u_y - u_{y_i})^T \bar{S}_{y,y_i}.$$

Finally, we define $z_j^{(i,w_j)} = \arg \max_{z_j \in H(x_i)} s(x_i, z_j, w_j)$ to be the best placement of part j on image x_i using the part filter w_j . We use this notation and rewrite $B_u(w, w^{old})$ in the following simple form:

$$B_u(w, w^{old}) = \lambda_w \|w\|^2 + \sum_{i=1}^k \max_{(y,z) \in Z_i} w^T \Phi(x_i, z, z^{(i,w^{old})}, a_{y,y_i}, \bar{a}_{y,y_i}) + \Delta(y_i, y). \quad (3.9)$$

We define a cache C to be a set of triplets (i, f, δ) where i indicates the i -th training example and f and δ indicate the feature vector $\Phi(x_i, z, z^{(i,w^{old})}, a_{y,y_i}, \bar{a}_{y,y_i})$ and the loss value $\Delta(y_i, y)$ associated to a particular $(y, z) \in Z_i$ respectively.

We denote the bound B_u subject to a cache C by \mathcal{B}_C and define it as follows:

$$\mathcal{B}_C(w) = \mathcal{B}_C(w; u, w^{old}) = \lambda_w \|w\|^2 + \sum_{i=1}^k \max_{(i,f,\delta) \in C} w^T f + \delta. \quad (3.10)$$

$\mathcal{B}_{C_A}(w) = B_u(w, w^{old})$ when C_A includes all possible latent configurations; that is

$$C_A = \{(i, f, \delta) | i \in \{1, \dots, k\}, f = \Phi(x_i, z, z^{(i,w^{old})}, a_{y,y_i}, \bar{a}_{y,y_i}), \delta = \Delta(y_i, y), (y, z) \in Z_i\}.$$

We denote the set of *hard* and *easy* examples of a dataset \mathcal{D} with respect to w

Algorithm 4 Fast optimization of $B_u(w, w^{old})$ using hard example mining

Input: w^{old}

- 1: $C_0 := \{(i, \mathbf{0}, 0) | 1 \leq i \leq k\}$
- 2: $t := 0$
- 3: **while** $\mathcal{H}(w^t, w^{old}, \mathcal{D}) \not\subseteq C_t$ **do**
- 4: $C_t := C_t \setminus \mathcal{E}(w^t, w^{old}, \mathcal{D})$
- 5: $C_t := C_t \cup \mathcal{H}(w^t, w^{old}, \mathcal{D})$
- 6: $w^{t+1} := \arg \min_w \mathcal{B}_{C_t}(w)$ $\triangleright \mathcal{B}_{C_t}$ defined as in Equation 3.10
- 7: $t := t + 1$
- 8: **end while**

Output: w^t

and w^{old} by $\mathcal{H}(w, w^{old}, \mathcal{D})$ and $\mathcal{E}(w, w^{old}, \mathcal{D})$, respectively, and define them as follows:

$$\begin{aligned} \mathcal{H}(w, w^{old}, \mathcal{D}) = & \{(i, \Phi(x_i, z, z^{(i, w^{old})}), a_{y, y_i}, \bar{a}_{y, y_i}), \Delta(y_i, y) | 1 \leq i \leq k, \\ & (y, z) = \arg \max_{(\hat{y}, \hat{z}) \in Z_i} w^T \Phi(x_i, \hat{z}, z^{(i, w^{old})}), a_{\hat{y}, y_i}, \bar{a}_{\hat{y}, y_i}) + \Delta(y_i, \hat{y})\} \end{aligned} \quad (3.11)$$

$$\begin{aligned} \mathcal{E}(w, w^{old}, \mathcal{D}) = & \{(i, \Phi(x_i, z, z^{(i, w^{old})}), a_{y, y_i}, \bar{a}_{y, y_i}), \Delta(y_i, y) | 1 \leq i \leq k, (y, z) \in Z_i, \\ & w^T \Phi(x_i, z, z^{(i, w^{old})}), a_{y, y_i}, \bar{a}_{y, y_i}) + \Delta(y_i, y) < 0\} \end{aligned} \quad (3.12)$$

We use the caching procedure outlined in Algorithm 4 to optimize the bound B_u . The algorithm starts with the initial cache $C_0 = \{(i, \mathbf{0}, 0) | 1 \leq i \leq k\}$ where $\mathbf{0}$ is the all-zero vector. This corresponds to the set of correct classification hypotheses; one for each training image. It then alternates between updating the cache and finding the part filters w^* that minimize \mathcal{B}_C . This is repeated until the cache remains unchanged. To update the cache we remove all the easy examples and add new hard examples to it (lines 4 and 5 of Algorithm 4). Note that $C_0 \subseteq C$ at all times.

Algorithm 4 may take up to 10 iterations to converge in practice, depending on the value of λ_w . However, one can save most of these cache-update rounds by retaining the cache content after convergence and using it to warm-start the next call to the algorithm. With this trick, except for the first call, Algorithm 4 takes only 2-3 rounds to converge. This is because many cache entries remain active even after w^{old} is updated; this happens, in particular, as we get close to the last iterations of the joint training loop (lines 2-8 of Algorithm 3). Note that to employ this trick one has

to modify the feature values (*i.e.* the f field in the triplets (i, f, δ)) of the entries in the retained cache according to the updated value of w^{old} .

The following theorem shows that the caching mechanism of Algorithm 4 works; meaning that it converges to $w^* = \arg \min_{w'} B_u(w', w^{old})$ for any value of u, w^{old} .

Theorem 3.1. *The caching mechanism converges to $w^* = \arg \min_{w'} B_u(w', w^{old})$.*

Proof. Let C_A be the cache that contains *all* possible latent configurations on \mathcal{D} . Assume that Algorithm 4 converges to w^\dagger after T iterations *i.e.* $w^\dagger = \arg \min_w \mathcal{B}_{C_T}(w)$. Since the algorithm converged $C_A \setminus C_T \subseteq \mathcal{E}(w^\dagger, w^{old}, \mathcal{D})$. Consider a small ball around w^\dagger such that for any w in this ball $\mathcal{H}(w, w^{old}, \mathcal{D}) \subseteq C_T$. The two functions $\mathcal{B}_{C_A}(w)$ and $B_u(w, w^{old})$ are equal in this ball and w^\dagger is a local minimum *inside* this region. $B_u(w, w^{old})$ is *strictly convex*, thus, w^\dagger is the global minimum of $B_u(w, w^{old})$.

To complete the proof we need to show that the algorithm does in fact converge. The number of possible caches is finite and the algorithm does not visit the same cache more than once, thus, it has to converge in a finite number of iterations. \square

Line 6 of Algorithm 4 is a convex optimization problem and can be solved using gradient based approaches or quadratic programming. We use the latter approach for the reasons mentioned in Section 1.5. This optimization problem becomes equivalent to that of (1.16) if we set $S_i = \{(\psi, \delta) | (i, f, \delta) \in C_t, \psi = f\}$.

3.6 Random Part Generation (1st Step of the Pipeline)

The joint training objective in (3.6) is non-convex making Algorithm 3 sensitive to initialization. Thus, the choice of initial parts can be crucial in training models that perform well in practice. We devote the first two steps of our pipeline to finding good initial parts (Figure 3-2). We then use those parts to initialize the joint training procedure of Section 3.5.

In the first step of our pipeline we randomly generate a large pool of initial parts. Generating a part involves picking a random training image (regardless of the image category labels) and extracting features from a random sub-window of the image

followed by whitening [26]. To whiten a feature vector f we use $\Sigma^{-1}(f - \mu)$ where μ and Σ are the mean and covariance of all patches in all training images. We estimate μ and Σ from 300,000 random patches. We use the norm of the whitened features to estimate discriminability of a patch. Patches with large whitened feature norm are farther from the mean of the background distribution in the whitened space and, hence, are expected to be more discriminative. Similar to [2] we discard the 50% least discriminant patches from each image prior to generating random parts.

Our experimental results with HOG features (Figure 3-4) show that randomly generated parts using the procedure described here perform better than or comparable to previous methods that are much more involved [31, 17, 63]. When using CNN features we get very good results using random parts alone, even before part-selection and training of the part filters (Figure 3-5).

3.7 Part Selection (2nd Step of the Pipeline)

Random part generation may produce redundant or useless parts. In the second step of our pipeline we train classifiers u using ℓ_1/ℓ_2 regularization (*a.k.a.* group lasso) to select a subset of parts from the initial random pool. This is similar to the procedure used in [63]. We group entries in each column of u . Let ρ_j denote the ℓ_2 -norm of the j -th column of u . The ℓ_1/ℓ_2 regularization is defined by $R_g(u) = \lambda \sum_{j=1}^m \rho_j$ where $\rho_j = \sqrt{\sum_y u_{y,j}^2}$ is the ℓ_2 -norm of the column of u that corresponds to part j .

If part j is not uninformative or redundant ρ_j (and therefore all entries in the j -th column of u) will be driven to zero by the regularizer. We train models using different values for λ to generate a target number of parts. The number of selected parts decreases monotonically as λ increases. Figure 3-3 in the supplement shows this. We found it important to retrain u after part selection using ℓ_2 regularization to obtain good classification performance.

Part selection is done by optimizing the following objective function:

$$\lambda \sum_{j=1}^m \rho_j + \sum_{i=1}^k \max\{0, \max_{y \neq y_i} (u_y - u_{y_i}) \cdot r(x_i, w) + 1\}. \quad (3.13)$$

This objective function is convex. We minimize it using stochastic gradient descent. This requires repeatedly taking a small step in the opposite direction of a sub-gradient of the function. Let $R_g(u) = \lambda \sum_{j=1}^m \rho_j$. The partial derivative $\frac{\partial R_g}{\partial u_y} = \frac{u_y}{\rho_j}$ explodes as ρ_j goes to zero. Thus, we round the ρ_j 's as they approach zero. We denote the rounded version by τ_j and define them as follows:

$$\tau_j = \begin{cases} \rho_j & \text{if } \rho_j > \epsilon \\ \frac{\rho_j^2}{2\epsilon} + \frac{\epsilon}{2} & \text{if } \rho_j \leq \epsilon \end{cases}.$$

The constants in the second case are set so that τ_j is continuous; that is $\frac{\rho_j^2}{2\epsilon} + \frac{\epsilon}{2} = \rho_j$ when $\rho_j = \epsilon$. In summary, part selection from an initial pool of parts $w = (w_1, \dots, w_m)$ involves optimizing the following objective function:

$$\lambda \sum_{j=1}^m \tau_j + \sum_{i=1}^k \max\{0, \max_{y \neq y_i} (u_y - u_{y_i}) \cdot r(x_i, w) + 1\} \quad (3.14)$$

We can control the sparsity of the solution to this optimization problem by changing the value of λ . In Figure 3-3 we plot ρ_j for all parts in decreasing order. Each curve corresponds to the result obtained with a different λ value. These plots suggest that the number of selected parts (*i.e.* parts whose ρ_j is larger than a threshold that depends on ϵ) decreases monotonically as λ increases. We adjust λ to obtain a target number of selected parts.

3.8 Experiments

We evaluate our methods on the MIT-indoor dataset [52]. We compare performance of models with randomly generated parts, selected parts, and jointly trained parts.

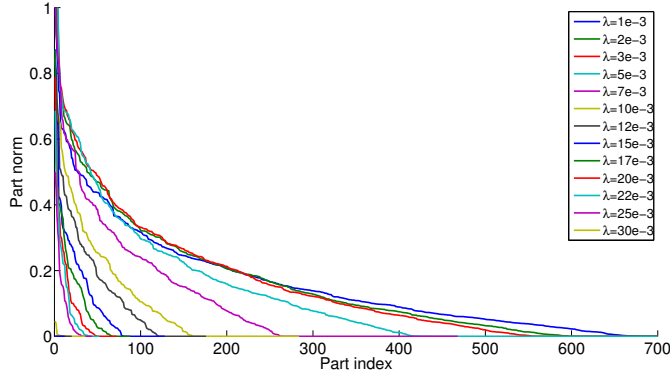


Figure 3-3: Effect of λ on part norms. Each plot shows sorted ρ_j values for a particular choice of λ .

We also compare performance of HOG and CNN features. The dataset has 67 indoor scene classes. There are about 80 training and 20 test images per class. Recent part-based methods that do well on this dataset [31, 17, 63] use a large number of parts (between 3350 and 13400).

HOG features: We resize images (maintaining aspect ratio) to have about $2.5M$ pixels, and extract 32-dimensional HOG features [14, 20] at multiple scales. Our HOG pyramid has 3 scales per octave. This yields about 11,000 patches per image. Each part filter w_j models a 6×6 grid of HOG features, so w_j and $\psi(x, z_j)$ are both 1152-dimensional.

CNN features: We extract CNN features at multiple scales from overlapping patches of fixed size 256×256 and with stride value $256/3 = 85$. We resize images (maintaining aspect ratio) to have about $5M$ pixels in the largest scale. We use a scale pyramid with 2 scales per octave. This yields about 1200 patches per image. We extract CNN features using Caffe [29] and the hybrid neural network from [73]. The hybrid network is pre-trained on images from ImageNet [16] and PLACES [73] datasets. We use the output of the 4096 units in the penultimate fully connected layer of the network (fc7). We denote these features by HP in our plots.

Part-based representation: Our final image representation is an mR -dimensional vector of part responses where m is the number of shared parts and R is the number of spatial pooling regions. We use $R = 5$ pooling regions arranged in a $1 \times 1 + 2 \times 2$ grid. To make the final representation invariant to horizontal image flips we average the mR -dimensional vector of part responses for image x and its right-to-left mirror

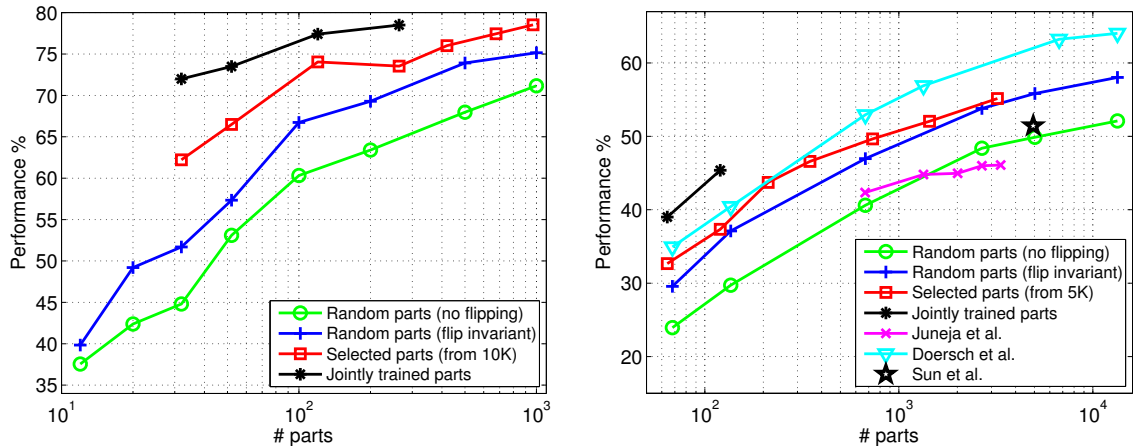


Figure 3-4: Performance of HOG features on 10-class subset (left) and full MIT-indoor dataset (right).

image x' to get $[r(x, w) + r(x', w)] / 2$ as in [17].

We first evaluate the performance of random parts. Given a pool of randomly initialized parts (Section 3.6), we train the part weights u using a standard ℓ_2 -regularized linear SVM; we then repeat the experiment by selecting few parts from a large pool using ℓ_1/ℓ_2 regularization (Section 3.6). Finally, we evaluate jointly trained parts (Section 3.5). While joint training significantly improves performance, it comes at a significantly increased computational cost making it impossible to do joint training on models with more than a few hundred shared parts.

Figure 3-4 shows performance of HOG features on the MIT-indoor dataset. HOG features are high dimensional (1152) and the number of potential placements in a HOG pyramid is huge (tens of thousands). This makes it prohibitively slow to do joint training with HOG features. To make it tractable, we consider training on a 10-class subset of the dataset for experiments with HOG features. The subset comprises *bookstore*, *bowling*, *closet*, *corridor*, *laundromat*, *library*, *nursery*, *shoeshop*, *staircase*, and *winecellar*. Performance of random parts increases as we use more parts. Flip invariance and part selection consistently improve results. Joint training improves the performance even further by a large margin achieving the same level of performance as the selected parts using much fewer parts. On the full dataset, random parts already outperform the results from [31], flip invariance boosts the performance beyond [63].

Joint training dominates other methods. However, we could not directly compare with the best performance of [17] due to the very large number of parts they use.

Figure 3-5 shows performance of CNN features on MIT-indoor dataset. As a baseline we extract CNN features from the entire image (after resizing to 256×256 pixels) and train a multi-class linear SVM. This obtains 72.3% average performance. This is a strong baseline. [55] get 58.4% using CNN trained on ImageNet. They improve the result to 69% after data augmentation.

We applied PCA on the 4096 dimensional CNN features to make them compact. This is necessary for making the joint training tractable both in terms of running time and memory footprint. Figure 3-5-left shows the effect of PCA dimensionality reduction. Surprisingly, we lose only 1% in accuracy with 160 PCA coefficients and only 3.5% with 60 PCA coefficients. We also show how performance changes when a random subset of dimensions is used. For joint training we use 60 PCA coefficients.

Figure 3-5-right shows performance of our part-based models using CNN features. For comparison with HOG features we also plot result of [17]. Note that part-based representation improves over the CNN extracted on the entire image. With 13400 random parts we get 77.1% (vs 72.3% for CNN on the entire image). The improvement is from 68.2% to 72.4% when we use 60 PCA coefficients. Interestingly, the 60 PCA coefficients perform better than the full CNN features when only a few parts are used (up to 1000). The gap increases as the number of parts decreases.

We do part selection and joint training using 60 PCA coefficients of the CNN features. We select parts from an initial pool of 1000 random parts. Part selection is most effective when a few parts are used. Joint training improves the quality of the selected parts. With only 372 jointly trained parts we obtain 73.3% classification performance which is even better than 13400 random parts (72.4%).

The significance of our results is two fold: 1) we demonstrate a very simple and fast to train pipeline for image classification using randomly generated parts; 2) we show that using part selection and joint training we can obtain similar or higher performance using much fewer parts. The gain is largest for CNN features ($13400/372 \approx 36$ times). This translates to 36x speed up in test time. See Section 3.9 for detailed

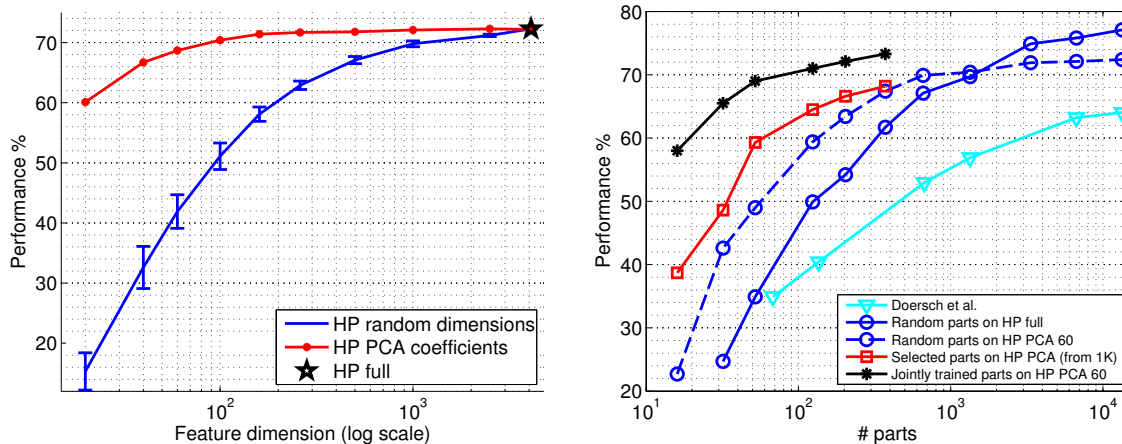


Figure 3-5: Performance of CNN features on the full MIT-indoor dataset. *HP* denotes the hybrid features from [73]. **Left:** the effect of dimensionality reduction on performance of the CNN features extracted from the entire image. Two approaches are compared; random selection over 5 trials (blue curve) and PCA (red curve). **Right:** part-based models with random parts (blue curves), selected parts from 1K random parts (red curve), and jointly trained parts (black curve).

run-time analysis of our method.

In Figure 3-4-left random parts (the blue curve) reach 73.9% using 500 parts while jointly trained parts (the black curve) reach 73.5% using 52 parts. In Figure 3-4-right random parts (the blue curve) reach 47.0% using 672 parts while jointly trained parts (the black curve) reach 45.4% using 120 parts.

3.8.1 Visualizing the Model Trained with CNN Features

Figure 3-6 shows the part weight matrix after joint training a model with 52 parts on the full MIT-indoor dataset. This model uses 60 PCA coefficients from the *HP* CNN features. Figure 3-7 shows top scoring patches for a few parts before and after joint training. The parts correspond to the model illustrated in Figure 3-6. The benefit of joint training is clear. The part detections are more consistent and “clean” after joint training. Part 17 seem to be detecting *cribs* initially but after joint training it becomes a more accurate *bed* detector while still firing on *cribs*.

Before joint training, majority of the detections of part 25 are *seats*. Joint training filters out the noise coming from *bed* and *sofa*. Part 46 consistently fires on *faces* even

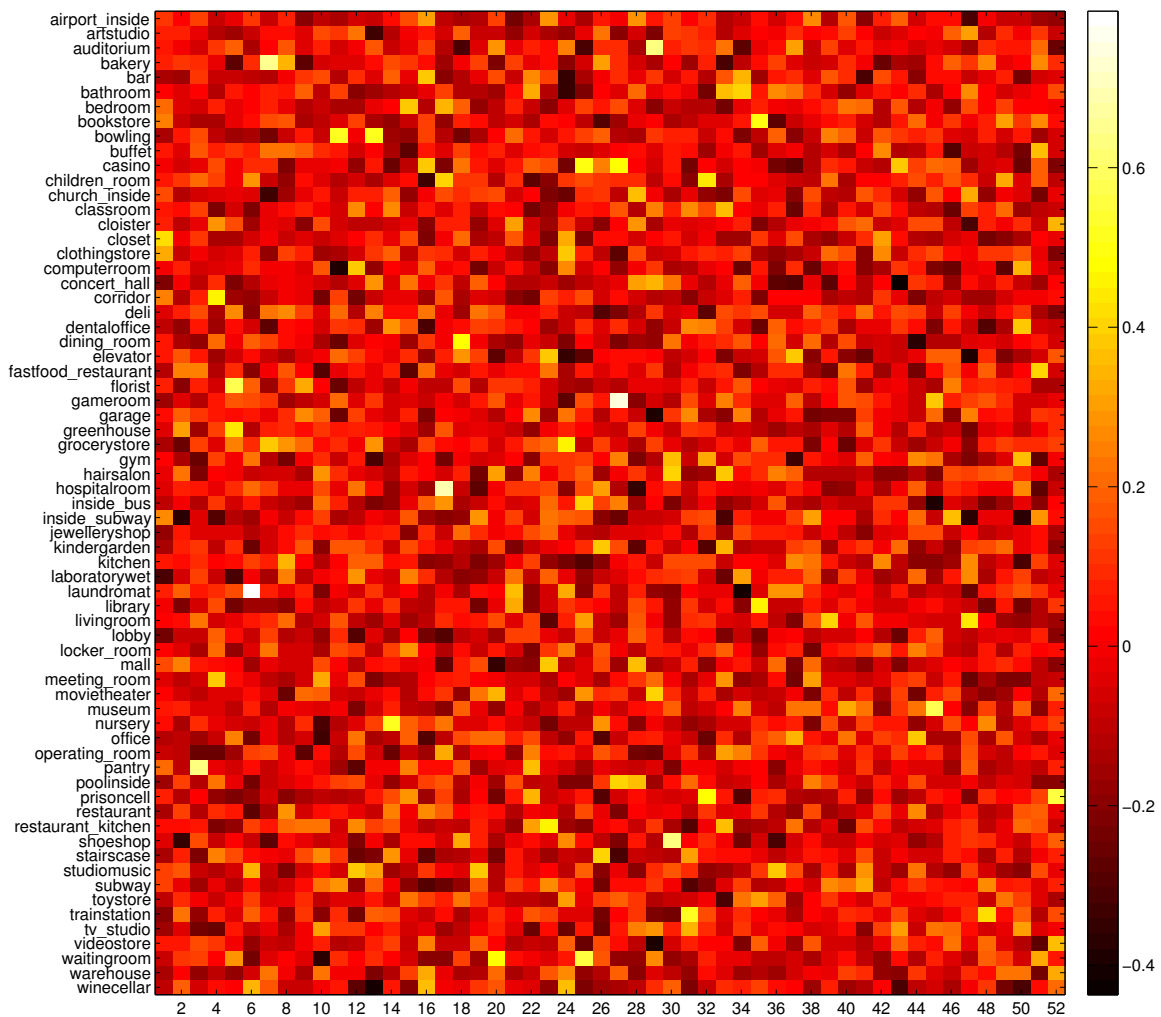


Figure 3-6: Part weights after joint training a model with 52 parts on the full dataset. Patches are represented using 60 PCA coefficients on CNN features. Although the model uses 5 pooling regions (corresponding to cells in $1 \times 1 + 2 \times 2$ grids) here we show the part weights only for the first pooling region corresponding to the entire image.

Top scoring patches on test images (multiple patches per image)



Figure 3-7: Top detections of two parts are shown before and after joint training on test images of the full MIT-indoor dataset. The numbers in the first column match the part indices in Figure 3-6.

before joint training. After joint training, however, the part becomes more selective to a single face and the detections become more localized.

Figure 3-8 illustrates selectivity of a few parts. Each row shows the highest scoring detections of a particular part on test images. The part indices in the first column match those of Figure 3-6. Even though most detections look consistently similar the images usually belong to multiple classes demonstrating part sharing across categories. For example, while part 17 appears to capture *bed* the images belong to *hospitalroom*, *childrensroom*, and *bedroom* classes. While part 25 appears to capture *seats* the images belong to *waitingroom*, *library*, *auditorium*, and *insidebus*. Conversely, multiple parts may capture the same semantic concept. For example, parts 3, 16, and 35 appear to capture *shelves* but they seem to be tuned specifically to shelves in *pantry*, *store*, and *book-shelves* respectively. Part 1 fires on *clothing-rack*, part 22 appear to find *container*, and part 33 detects *table-top*. Some parts respond to a part of an object; *e.g.* part 40 and 46 respond to *leg* and *face*. Others find entire objects or even composition of multiple objects. For example, parts 6, 17, 37, 43 detect *laundromats*, *beds*, *cabinets*, and *monitor*. Part 29 detects composition of *seats-and-screen*. There are also parts that capture low-level features such as the mesh pattern of part 31 and the high-frequency horizontal stripes of part 41. Also, there are parts that are selective for certain colors. For example, part 9 appears to be capturing specific *red* patterns (in particular *fruits* and *flowers*). Part 48 is very well tuned to finding *text*.

The part weight matrix u (Figure 3-6) helps us better understand how parts assists classification. Part 6 has significantly high weight for class *laundromat* and it appears to be a good detector thereof. Part 27 fires strongly on game/sports-related scenes. The weight matrix reveals that this part is strongly correlated with *gameroom*, *casino*, and *poolinside*. Part 17 fires strongly on *bed* and it has the highest weight for *hospitalroom*, *children_room*, *bedroom*, and *operating_room*.

The weight matrix also identifies negative parts. An interesting example is part 46 (the face detector). One would expect this part to be positive for most classes, reflecting presence of humans in indoor scenes. Part 46 has the lowest weight for *buffet*, *classroom*, *computerroom*, and *garage*, suggesting that it is a negative part for

these classes relative to others. This is surprising because one would expect to find people in scenes such as *classroom* and *computerroom*. We examined all training images of these classes and found no visible faces in them except for 1 image in *classroom* and 3 images in *computerroom* with hardly visible faces and 1 image in *garage* with a clear face in it.

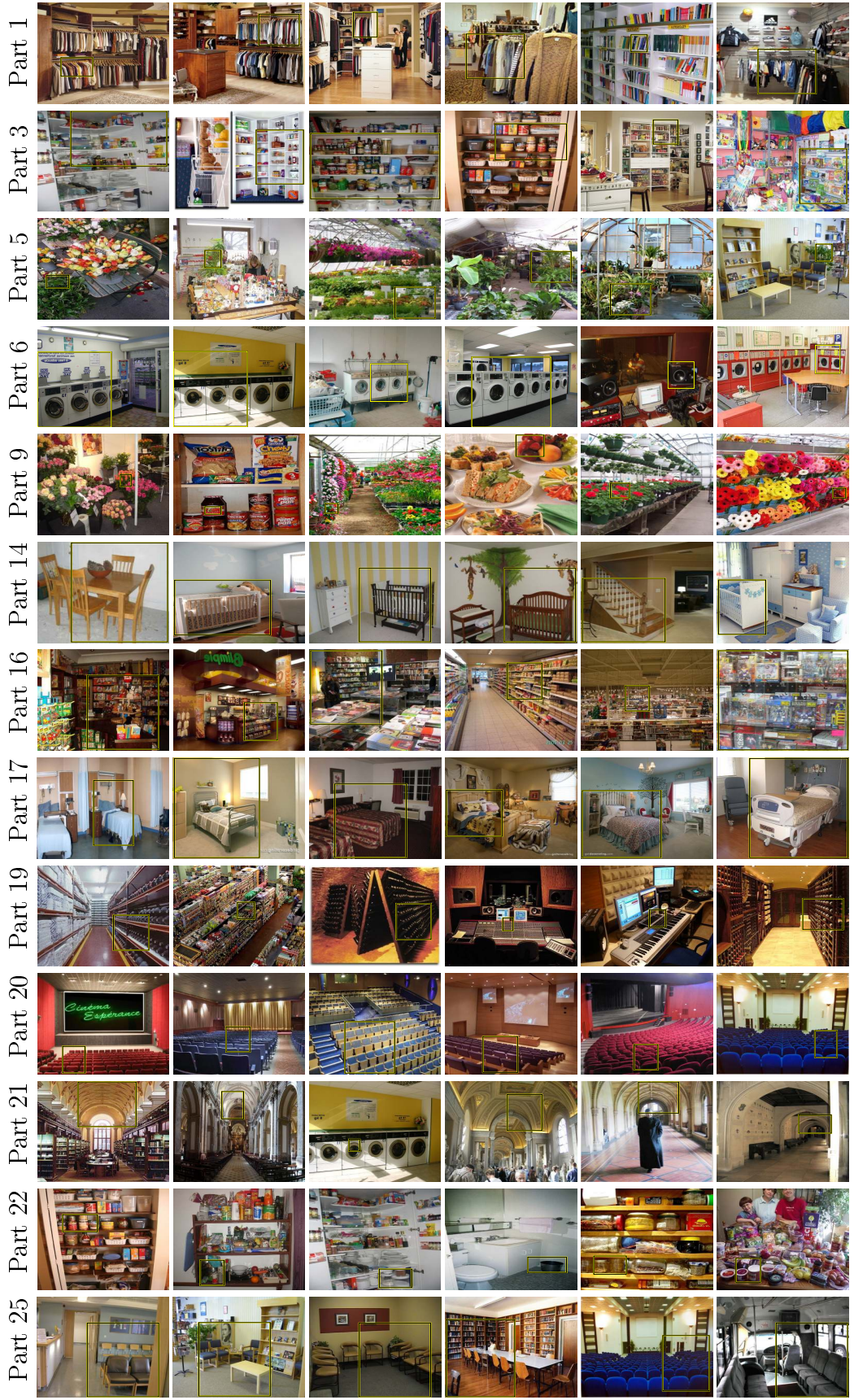
Part 21 is highly weighted for *laundromat*, *library*, and *cloister* and it appears to respond strongly to *arch*. Also note that part 21 is a strong negative part for *bookstore* relative to *library*. Presence of an *arch*, in fact, is a very sensible differentiating pattern that could tell *library* apart from *bookstore*.

3.8.2 Visualizing the Model Trained with HOG Features

In this section we visualize models trained using HOG features [14]. Figure 3-9 shows the part filters and the weight matrix after joint training a model with 52 parts on the 10-class subset of MIT-indoor dataset. The part weight matrix illustrates whether a part is positive or negative with respect to two categories. For example, part 42 is positive for *bookstore* and *library* relative to *laundromat*. Part 29 is positive for *laundromat* relative to *bookstore* and *library*. Part 37 is positive for *library* relative to *bookstore* so it can be used in combination with the other two parts to distinguish between all three categories *bookstore*, *library*, and *laundromat*. Figure 3-10 illustrates the top scoring patches for these three parts.

3.9 Processing Time

Test time: the test procedure of our models involves three simple steps: 1) convolving part filters with the test image, 2) computing the part-based representation 3) finding the class with the highest classification score. Step 1 takes $O(mhd)$ time where m , h , and d are the number of parts, latent locations, and dimensionality of the patch features. Step 2 takes $O(hR)$ time where R is the number of pooling regions. Step 3 takes $O(nmR)$ time where n is the number of classes. The bottleneck in test time is step 1 and 3 both of which depend on the number of parts m . So, a decrease in



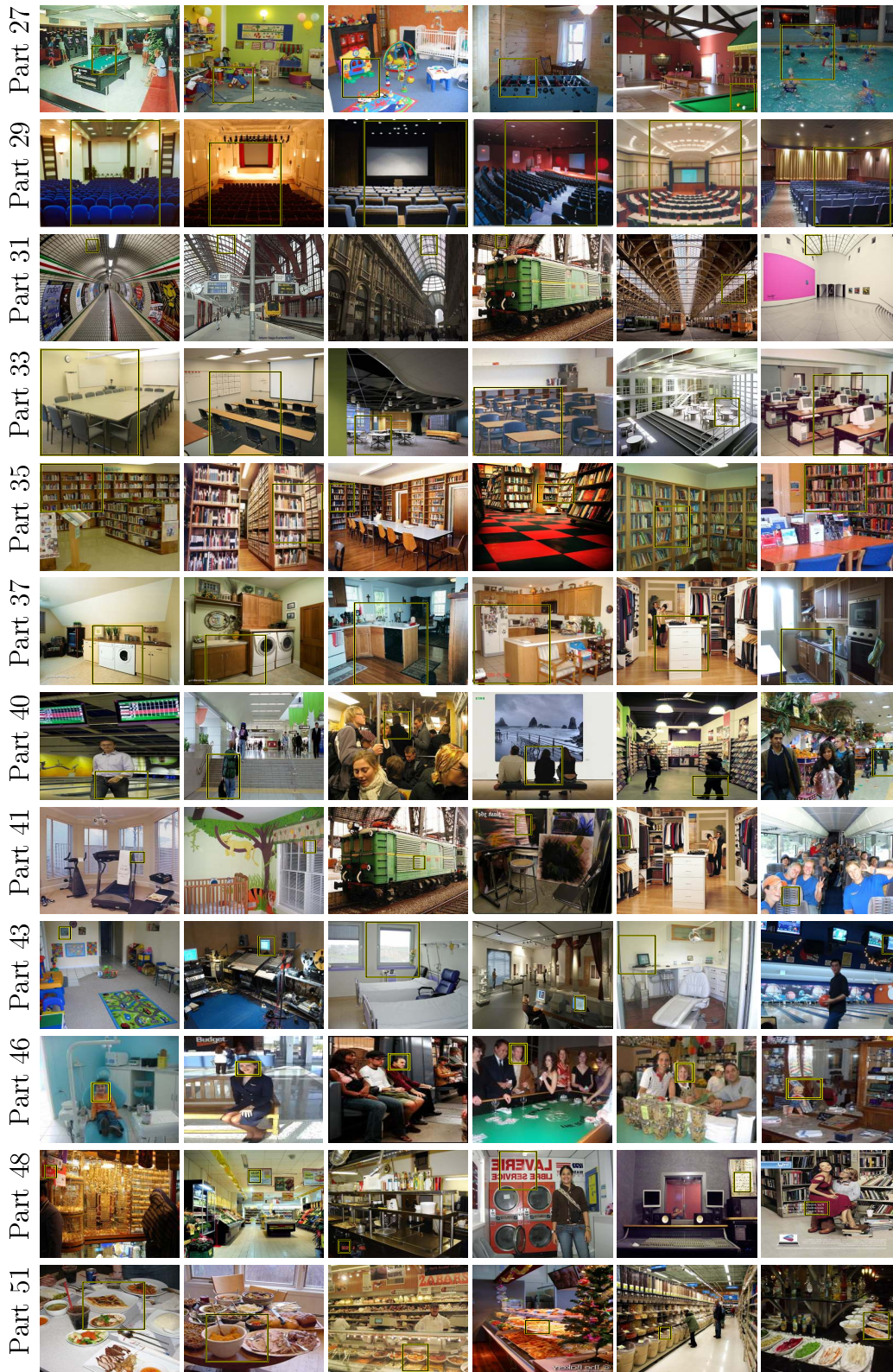


Figure 3-8: Top detections of parts on test images of the full dataset. The numbers in the first column match the part indices in Figure 3-6. Part detection is done in a multi-scale sliding window fashion and using a 256×256 window. For visualization purposes images are stretched to have the same size.

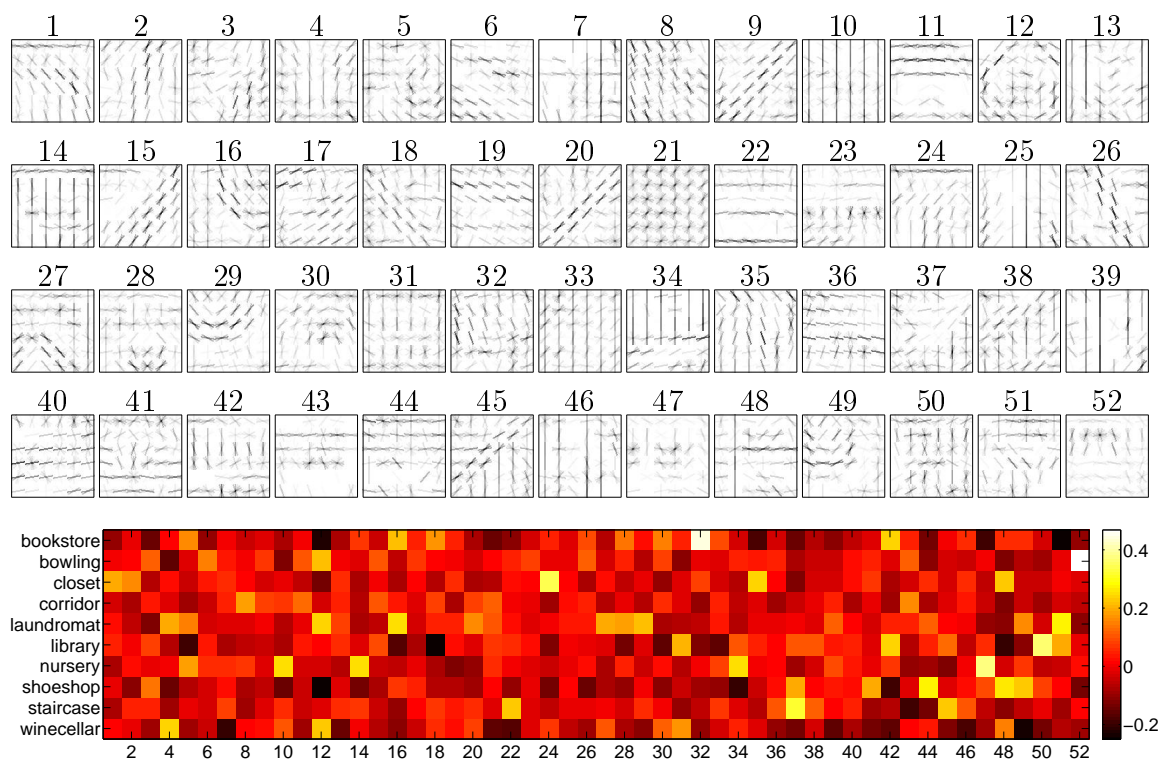


Figure 3-9: Part filters (top) and part weights (bottom) after joint training a model with 52 parts on the 10-class dataset. Here we use HOG features. Although the model uses 5 pooling regions (corresponding to cells in $1 \times 1 + 2 \times 2$ grids) here we show the part weights only for the first pooling region corresponding the entire image.

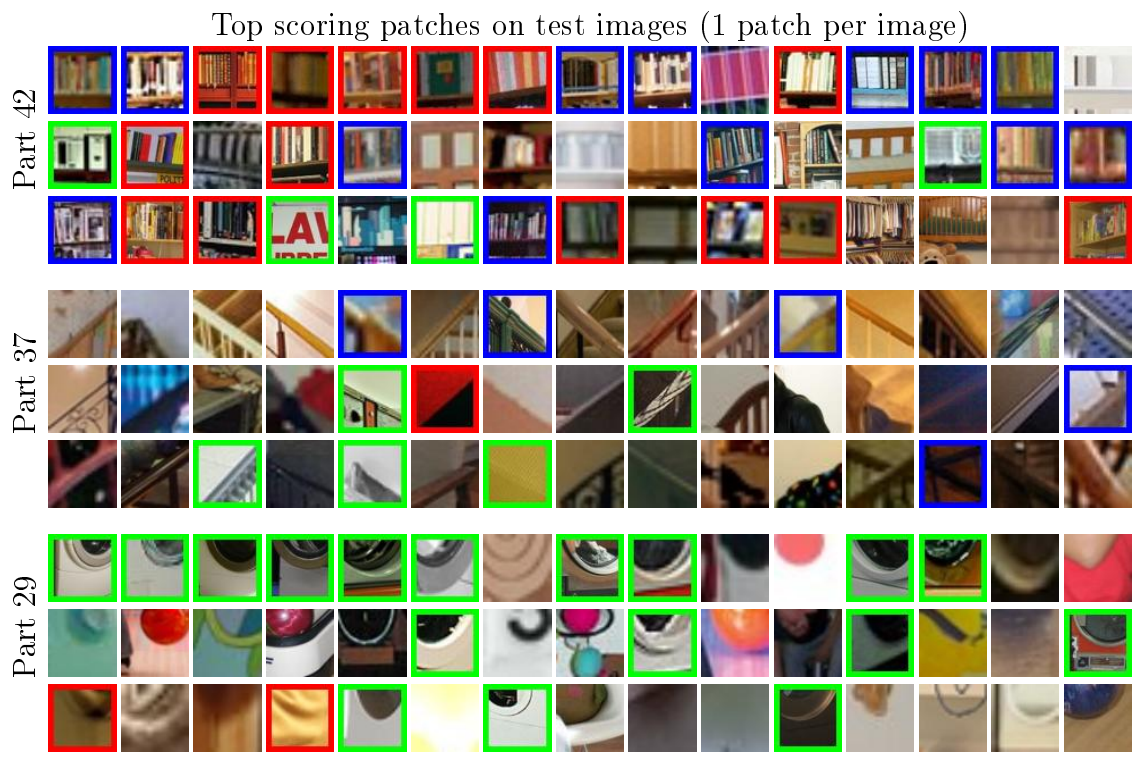


Figure 3-10: Top detections of three parts on test images of the 10-class dataset. The numbers in the first column match the part indices in Figure 3-9. Patches from *bookstore*, *laundromat*, and *library* images are highlighted in red, green, and blue respectively (best viewed in color).

m directly affects the test time. Note that both of these two steps are embarrassingly parallel processes.

Training time: the training procedure involves two main steps: 1) learning part weights (line 3 in Algorithm 3) and 2) learning part filters (lines 4-7 in Algorithm 3). The first step is a standard multi-class SVM problem and is relatively fast to train. The bottleneck in training is the second step.

Learning part filters involves multiple nested loops: 1) joint training loop (lines 2-8 in Algorithm 3), 2) relabeling loop (lines 4-7 in Algorithm 3), 3) cache update loop (lines 4-9 in Algorithm 4), and 4) the constraint generation loop of the QP solver (lines 3-10 in Algorithm 1). The number of iterations each loop takes depends on the training data and the hyper parameters of the model (*i.e.* λ_w and λ_u).

We report the running time of our joint training algorithm separately for one experiment that uses HOG features and one that uses CNN features as the dimensionality of the features and the number of latent locations they consider is different.

In our current implementation it takes 5 days to do joint training with 120 shared parts on the full MIT-indoor dataset on a 16-core machine using HOG features. It takes 2.5 days to do joint training with 372 parts on the full dataset on a 8 core machine using 60-dimensional PCA-reduced CNN features. Note that these time include learning all shared part filters and all 67 class-specific part weight vectors *on a single machine*. In both experiments finding the most violated constraint (line 8 in Algorithm 1) takes more than half of the total running time. The second bottleneck for HOG features is growing the caches (line 6 in Algorithm 4). This involves convolving the part filters (1152 dimensional HOG templates) with all training images (each containing 11000 candidate locations). With the CNN features, however, the second bottleneck becomes the QP solver (line 7 in Algorithm 4). The QP solver that we use only uses a single core. In both cases the ratio of the time taken by the first bottleneck to the second one is 4 to 1.

The pipeline in previous methods such as [31, 63, 17] has several steps. For example, to discover parts, [31] applies multiple superpixel segmentations on the image to find initial seeds, trains exemplar LDA for each seed, enhances the candidate

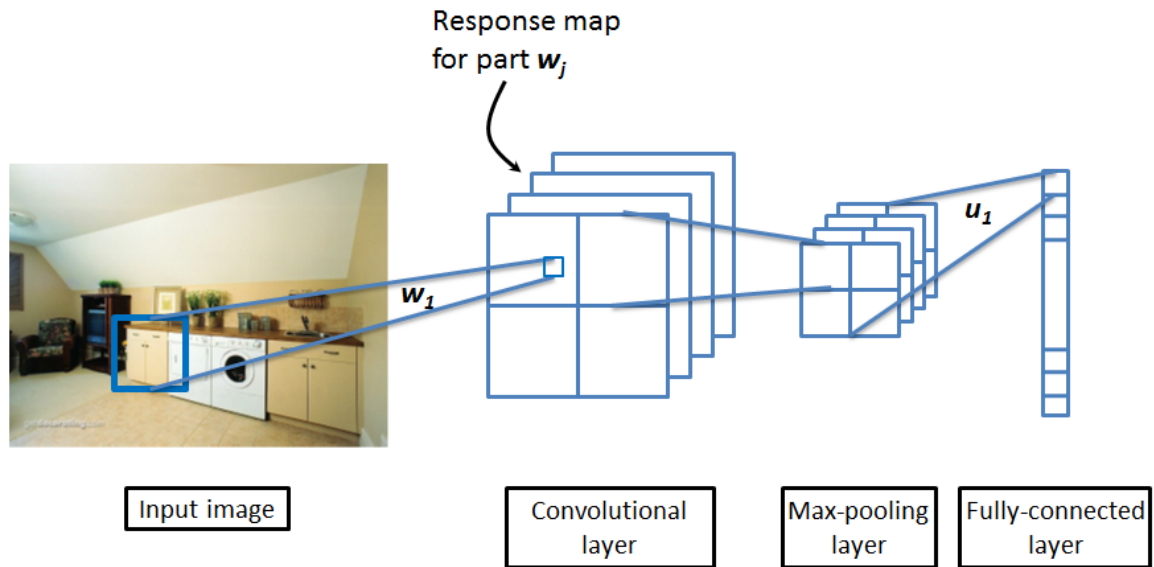


Figure 3-11: Our part-based model can be thought of as a 3-layer Neural Network. The first layer is convolutional. Different channels correspond to different parts. The second layer performs the max-pooling operation in 4 quadrants of the part response maps. The last layer is fully connected and performs 67-way classification.

parts by harvesting similar patches in the dataset, and computes the entropy of the top-50 detections of each part over categories. They discard parts with high entropy as well as duplicates. Besides using several heuristics, these methods are also slow; [17] do not comment on the processing time of their method but we know from personal correspondence that their code takes a long time to run. However, most of the steps in their method are independent; e.g. they start their method from multiple initial points to find the discriminative modes, they train 1-vs-all classifiers, etc. So, they distribute the processing load on a big cluster in order to run their experiments.

Our experimental results showed that we can obtain better performance than [31] and [63] using a pool of randomly initialized parts (see Figure 3-4). Note that creating a pool of random parts is very straightforward and fast. It only takes extracting features from random sub-window of a random image and applying a simple feature transformation on them (see Section 3.6).

3.10 Connection to Convolutional Neural Networks

The part-based model that we presented in this chapter can be implemented as a convolution neural network with three layers. This is illustrated in Figure 3-11. The first layer convolves part filters with the input image (or some features extracted from it). Each part produces a response map independently from the others. The second layer performs the max-pooling operation. And the third (fully connected) layer performs the classification.

Convolutional Neural networks are typically trained using the back-propagation algorithm. Success of back-propagation hinges upon the selection of a good learning rate schedule. The learning rate schedules, however, are application dependent and hard to tune. Moreover, in practice, a stochastic version of the back-propagation algorithm is used which makes the training even less reliable and predictable.

We, on the other hand use sequential convex optimization for training which is more predictable, robust, and reliable than back-propagation. More specifically, our optimization algorithm is guaranteed to (1) improve the training objective in each step, and (2) converge to a local minimum or saddle point of the objective function.

3.11 Conclusion

We presented a simple pipeline to train part-based models for image classification. All model parameters are trained jointly in our framework; this includes shared part filters and class-specific part weights. All stages of our training pipeline are driven *directly* by the same objective namely the classification performance on a training set. In particular, our framework does not rely on ad-hoc heuristics for selecting discriminative and/or diverse parts. We also introduced the concept of “negative parts” for part-based models.

Models based on our randomly generated parts perform better than almost all previously published work despite the profound simplicity of the method. Using CNN features and random parts we obtain 77.1% accuracy on the MIT-indoor dataset,

improving the state-of-the-art. We also showed that part selection and joint training can be used to train a model that achieves better or the same level of performance as a system with randomly generated parts while using much fewer parts.

Joint training alternates between training part weights and updating part filters. This process can be initiated before the first or the second step leading to two different initialization schemes. Currently we use random examples to initialize the part filters. It would also be possible to initialize the entries in u based on how a hypothetical part is correlated with a class; negatively, irrelevant, or positively. Training the part filters would then learn part models that fit this description.

Our part-based models can be seen as 3-layer convolutional neural networks. ConvNets are typically trained using stochastic back-propagation which is hard to tune in practice. Our training algorithm, on the other hand, uses sequential convex optimization instead which is more robust and reliable in practice.

Chapter 4

Generalized Latent Variable Models

Discriminative latent variable models (LVM) are frequently and successfully applied to various visual recognition tasks [20, 24, 70, 54, 62, 47, 48]. Latent variables provide a formalism for modeling structured variation of visual features. Conventionally, latent variables are defined on the variation of the foreground (positive) class. We augment LVMs to include *negative* latent variables corresponding to the background class. Negative latent variables can, for instance, learn mutual exclusion constraints, model scene subcategories where the positive object class is unlikely to be found, or capture specific parts that indicate the presence of an object of a similar but different class.

We formulate the family of Generalized Latent Variable Models (GLVMs) in Section 4.1. We propose a procedure for learning parameters in GLVMs in Section 4.2. In Section 4.3 we show how the training algorithm of the well known Deformable Part Models of [20] changes when we add negative latent variables (or parts) to it. Finally, in Section 4.4 we study the performance of such generalizations of DPMs and demonstrate considerable improvement on two detection datasets.

4.1 Generalized Latent Variable Models (GLVMs)

In binary classification inputs are classified into *foreground* ($y = +1$) and *background* ($y = -1$). In this chapter we focus on binary classification problems which suit many real-world applications better, compared to multi-class classification. We generalize

LVMs by equipping them with latent variables that look for “counter evidence” for the foreground class. For example, for a *cow* detector, presence of saddle/ rider/aeroplane counts as counter evidence whereas presence of meadow/stable counts as evidence.

We call the latent variables that collect evidence for the foreground class *positive latent variable* and the latent variables that collect counter evidence for the foreground class *negative latent variable* and denote them by a and b respectively throughout this chapter. Note that this is related to the concept of positive and negative parts that we discussed earlier in Chapter 3 (see Definition 3.1 in Section 3.3).

In binary classification we are interested in the relative score of the foreground class compared to the background class. In particular, we only need to model the score difference between foreground and background and classification can be done by comparing the value of the difference to a fixed threshold. So, the score function of a 2-class LVM can be written as follows (*cf.* Equation 1.27):

$$f_w(x_i) = \max_{a \in Z} w \cdot \phi(x_i, a). \quad (4.1)$$

We define the simplest form of a GLVM score function involving a positive latent variable $a \in Z^+$ and a negative latent variable $b \in Z^-$ as follows:

$$f_w(x_i) = \max_{a \in Z^+} w \cdot \phi(x_i, a) - \max_{b \in Z^-} w \cdot \phi(x_i, b) \quad (4.2)$$

$$\begin{aligned} &= \max_{a \in Z^+} w \cdot \phi(x_i, a) + \min_{b \in Z^-} -w \cdot \phi(x_i, b) \\ &= \max_{a \in Z^+} \min_{b \in Z^-} w \cdot \phi(x_i, a) - w \cdot \phi(x_i, b) \\ &= \max_{a \in Z^+} \min_{b \in Z^-} w \cdot \phi(x_i, a, b), \end{aligned} \quad (4.3)$$

where $\phi(x_i, a, b) = \phi(x_i, a) - \phi(x_i, b)$. Though this assumes that a and b are independent, in general, $\phi(x_i, a, b)$ can be any function of the joint latent configuration (a, b) . One can show that when a and b are independent the training objective that uses (4.2) is equivalent to a 2-class LS-SVM. We also note that GLVM is more general than LVM for binary classifiers, in particular, (4.2) reduces to (4.1) when $Z^- = \emptyset$.

The score associated to a positive latent variable (*i.e.* $\max_{a \in Z^+} w \cdot \phi(x_i, a)$) is positively correlated with the final classification score. The score associated to a negative latent variable, however, is negatively correlated with the final classification score. This is not to be confused with negative entries in the model vector w . In a model with linear score function $f_w(x) = w \cdot \phi(x)$ the value of the d -th dimension of the feature vector (*i.e.* $\phi(x)_d$) is negatively correlated with $f_w(x)$ when $w_d < 0$. Yet, negative latent variables are fundamentally different from this. In particular, a negative latent variable is not equivalent to flipping the sign in some dimensions of the parameter vector and using a positive latent variable instead.

We can extend the score function of Equation 4.3 by making a hierarchy of alternating positive and negative latent variables recursively. This general formulation has interesting ties to compositional models, like object grammar [21, 22], and high-level scene understanding. For example, we can model scenes according to the presence of some objects (positive latent variable) and the absence of some other objects (negative latent variable). Each of those objects themselves can be modeled according to presence of some parts (positive) and absence of others (negative). We can continue the recursion further. The GLVM score function, after being expanded recursively, can be written in the following general form:

$$f_w(x_i) = \max_{a_1} \min_{b_1} \max_{a_2} \min_{b_2} \dots \max_{a_K} \min_{b_K} w \cdot \phi(x_i, (a_k, b_k)_{k=1}^K) \quad (4.4)$$

We call (4.4) the *canonical form* of a GLVM score function. Note that any other score function that is a linear combination of *max*'s and *min*'s of linear functions can be converted to this canonical form.

4.2 Training GLVMs

In this section we propose an algorithm for training GLVMs to do binary classification. We focus on SVM training objective and use hinge loss. However, the same idea can be used to optimize any training objective that involves a linear combination of convex

and/or concave functions of GLVM scores. For the ease of notation we drop the subscript w in f_w in the rest of this section.

Let $D = \{(x_i, y_i)\}_{i=1}^n$ denote a set of n labeled training examples. Each example is a pair (x_i, y_i) where x_i is the input data and $y_i \in \{+1, -1\}$ is the ground-truth label associated with it. The training objective for binary SVM can be written as:

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max \{0, 1 - y_i f(x_i)\}. \quad (4.5)$$

We train GLVMs using an iterative algorithm that is very similar to CCCP [72]. But, unlike CCCP, we do not linearize the concave part of the objective function. Our training algorithm alternates between two steps: 1) constructing a convex upper bound to the training objective O and 2) minimizing the bound. Let $b_t(w)$ denote the bound constructed at iteration t . Also, let $w_t = \arg \min_w b_t(w)$ be the minimizer of the bound. All bounds touch the objective function at the solution of the previous bound; that is $b_t(w_{t-1}) = O(w_{t-1}), \forall t$. This process is guaranteed to converge to a local minimum or a saddle point of the training objective O .

To construct bounds we only need to construct a concave lower bound $\hat{f}(x_i)$ and a convex upper bound $\check{f}(x_i)$ to the score function of (4.4). We define b_t as follows:

$$b_t(w) = \frac{1}{2} \|w\|^2 + C \sum_{\substack{i=1 \\ y_i=-1}}^n \max \{0, 1 + \check{f}_t(x_i)\} + C \sum_{\substack{i=1 \\ y_i=+1}}^n \max \{0, 1 - \hat{f}_t(x_i)\}. \quad (4.6)$$

In the next section we will explain how $\check{f}_t(x_i)$ and $\hat{f}_t(x_i)$ are constructed for GLVMs. Algorithm 5 summarizes all steps in our training procedure.

4.2.1 Convex Upper Bound and Concave Lower Bound

To obtain a convex upper bound $\check{f}_t(x_i)$ and a concave lower bound $\hat{f}_t(x_i)$ on the GLVM score function of (4.4) we fix all the negative and all the positive latent variables in $f(x_i)$, respectively.

Let w_t denote the model at iteration t , and $a = (a_1, \dots, a_K)$ be an arbitrary

Algorithm 5 Training GLVM classifiers.

Input: $D = \{(x_i, y_i)\}_{i=1}^n, w_0$

1: $t \leftarrow 0$

2: **repeat**

3: $t \leftarrow t + 1$

4: Construct $\check{f}_t(x)$

▷ from Equation 4.9 and w_t

5: Construct $\hat{f}_t(x)$

▷ from Equation 4.10 and w_t

6: Construct $b_t(w)$

▷ from Equation 4.6

7: $w_t \leftarrow \arg \min_w b_t(w)$

8: **until** w_t does not change

Output: w_t

assignment of the positive latent variables. Given a , we denote the fixed values for the negative latent variables on image x_i by $b^{(t)}(x_i, a) = (b_1^{(t)}(x_i, a), \dots, b_K^{(t)}(x_i, a))$ and define it in (4.7). Similarly, we define $a^{(t)}(x_i, b) = (a_1^{(t)}(x_i, b), \dots, a_K^{(t)}(x_i, b))$ in (4.8).

$$b^{(t)}(x_i, a) = \arg \min_{b_1, \dots, b_K} w_t \cdot \phi(x_i, (a_k, b_k)_{k=1}^K) \quad (4.7)$$

$$a^{(t)}(x_i, b) = \arg \max_{a_1, \dots, a_K} w_t \cdot \phi(x_i, (a_k, b_k)_{k=1}^K) \quad (4.8)$$

Note that (4.7) uses a min and (4.8) uses a max. Also, note that both equations use w_t to compute the fixed assignments. Finally, we define $\check{f}_t(x_i)$ and $\hat{f}_t(x_i)$ as follows:

$$\check{f}_t(x_i) = \max_{a_1} \max_{a_2} \dots \max_{a_K} w \cdot \phi\left(x_i, \left(a_k, b_k^{(t-1)}(x_i, a)\right)_{k=1}^K\right), \quad (4.9)$$

$$\hat{f}_t(x_i) = \min_{b_1} \min_{b_2} \dots \min_{b_K} w \cdot \phi\left(x_i, \left(a_k^{(t-1)}(x_i, b), b_k\right)_{k=1}^K\right). \quad (4.10)$$

Equation 4.9 is convex because it is a max of linear functions and it is an upper bound of $f(x)$ because the min functions are replaced with linear functions. Similarly, Equation 4.10 is a concave lower bound of $f(x)$.

4.3 Adding Negative Parts to DPMs

Part based models have had a long and successful presence in the history of Computer Vision [20, 12, 36, 8, 77]. The Deformable Part Models (DPMs) of [20], in particular,

have been extensively used for object detection over the past decade. Despite their incredible flexibility to object deformation and viewpoint changes, DPMs only look for positive evidence in images and do not capture counter evidence. For example, the knowledge that a *saddle* detector fires strongly on a detection window should decrease the score of *cow* detector on that window. This is because *cows* tend not to have *saddles* on their backs. Thus, if we find a *saddle* in the detection window of a *cow* detector we, most likely, are confusing a *horse* for a *cow*. Counter evidence is particularly important in discriminating between similar objects; like *cow* from *horse* in the hypothetical example above. We obtain this behavior by augmenting *cow* detector model with a negative part that fires on *saddle*.

A DPM is a mixture of multi-scale part-based detectors that model the appearance of an object in different viewpoints. Each mixture component uses a root filter to capture the global structure of the object in a coarse scale, and a set of *deformable* part filters to capture the appearance of parts of the object in a higher resolution.

DPM is a latent variable model and, as such, uses a score function of the form in (4.1) where the latent variable a captures the choice of the mixture component (which we denote by c), the location and scale of the root filter (which we denote by l_0), and the location of the parts (which we denote by l_1, \dots, l_m for a model with m parts), and we have $a = (c, l_0, l_1, \dots, l_m)$.

4.3.1 Training DPMs without Negative Parts

Let $\mathcal{D} = \{(x_i, y_i, box_i)\}_{i=1}^n$ denote a set of n annotated training examples where x_i is an image, $y_i \in \{+1, -1\}$ is a class label that determines whether the sample belongs to the foreground class ($y_i = +1$) or the background class ($y_i = -1$), and box_i specifies the coordinates of an annotated bounding box that circumscribes an instance of the object in a foreground image. In background images any bounding box is an independent negative example, and thus box_i is irrelevant. DPM uses the root filter as a detection window during training. Despite availability of *boxes* on foreground images, any placement of the root filter whose overlaps with an annotated bounding box box_i exceeds a certain threshold τ is considered as a potential true positive for

box_i . Let $H(x_i, box_i, \tau)$ denote the set of all valid detection windows for box_i . DPM uses a latent variable to choose the true detection window from this set.

In order to train a DPM we first construct a training set D from the annotated examples in \mathcal{D} . Each element in D is a triplet (x_i, y_i, R_i) where x_i is an image, y_i is the class label, and R_i specifies a set of valid locations for placing the root filter of the DPM. Each background example $(x, y = -1, box) \in \mathcal{D}$ has multiple examples $(x, y, \{r\})$ associated to it in D , one for each possible root placement r . Each foreground example $(x, y = +1, box) \in \mathcal{D}$ has exactly one example (x, y, R) associated to it in D where $R = H(x, box, \tau)$. In DPM, the value of τ is set to 0.7. The training objective of DPM with k mixture components can be written as follows:

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \max \left\{ 0, 1 - y_i \max_{l_0 \in R_i} \max_{c=1}^k \max_{l_1, \dots, l_m} w \cdot \phi(x_i, c, l_0, l_1, \dots, l_m) \right\}. \quad (4.11)$$

Note that D is populated with numerous negative examples per each background image of \mathcal{D} and therefore $N \gg n$, where N denotes the size of D and n denotes the size of \mathcal{D} . This makes the direct optimization of the objective function of (4.11) prohibitively slow. In practice, DPM uses a data mining procedure to minimize (4.11). This involves maintaining a small subset of the training examples in D together with a few latent configurations for each example, and optimizing the training objective on this reduced set, *a.k.a.* the *cache*. Each entry in the cache stores a latent configuration $a = (c, l_0, l_1, \dots, l_m)$ and an image index i from which one can compute the loss value associated to the cache entry for a given w as $\max \{0, 1 - y_i w \cdot \phi(x_i, c, l_0, l_1, \dots, l_m)\}$. The loss value associated to an example x_i *with respect to the cache* is the maximum loss of all the entries in the cache whose image index is i .

DPM deploys CCCP for training. In each iteration of CCCP the latent variables (*i.e.* c, l_0, l_1, \dots, l_m) for all *positive* examples in D are fixed. Initially, none of the entries in the cache correspond to negative examples, and the cache only contains one entry for each positive training example corresponding to the latent configuration that was fixed for it. The data mining procedure alternates between optimizing the training

objective with respect to the current cache and then updating the content of the cache. The cache-update step removes entries, corresponding to negative examples, that score lower than -1 on the current model (*a.k.a. easy negatives*), and then adds a new entry per negative example to the cache corresponding to the highest scoring latent configurations of the negative example provided that it scores at least -1 on the current model (*a.k.a. hard negatives*), while making sure there are no duplicate entries in the cache. It is guaranteed that the data mining procedure stops after a finite number of iterations and the minimizer of the cache converges to the minimizer of the training objective of (4.11). In practice, the size of the cache remains small at all times¹, making the optimization of the objective function on the cache fast.

4.3.2 Training DPMs with Negative Parts

We extend DPMs by adding negative parts to them. We call the extended model Generalized DPM (GDPM). The objective function of a GDPM with m positive parts and q negative parts is as follows:

$$O(w) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \max \left\{ 0, 1 - y_i \max_{l_0 \in R_i} \max_{c=1}^k \max_{l^+} \min_{l^-} w \cdot \phi(x_i, c, l_0, l^+, l^-) \right\}, \quad (4.12)$$

where the latent variables $l^+ = (l_1^+, \dots, l_m^+)$ and $l^- = (l_1^-, \dots, l_q^-)$ capture the location of positive and negative parts respectively. Note that, we maximize over all latent variables except for l^- , which we minimize over.

To minimize the training objective of (4.12) we need to construct a concave lower bound on the score of the positive samples and a convex upper bound on the score of the negative samples (see Section 4.2.1 for details). This can be done by fixing the location of the negative parts for the negative samples (as in (4.9)), and the location of the positive parts, the location of the root filter, and the mixture component for

¹DPM stops growing the cache if the size of the cache reaches a pre-defined limit. This can happen in the early stages of training, when the model is far from optimal. However, the limit is set high enough that it never stalls converges of the cache.

positive samples (as in (4.10)). A naive way of doing this is to store the location of the fixed latent variables for any configuration of the free latent variables. For example, for a negative image, this requires storing the location of the negative parts *i.e.* l^- for any given placement of the root filter l_0 , the positive parts l^+ , and selection of the mixture component c . This requires a prohibitively large amount of memory. To do this more efficiently, one can use the fact that the dependency structure in a DPM follows a star model, and all parts become independent conditional on the root filter location and the choice of the mixture component. This also requires an extremely large amount of memory and is not practical.

We propose a memory-efficient way to construct convex upper bounds and concave lower bounds to the score function by saving a copy of the model from the previous iteration and *inferring* the value of the fixed latent variables using the stored model. Although this entails an added computational cost of the inference process, but, has a very small memory footprint (it only requires saving a copy of the model parameters). More precisely, for a negative example, we use the stored model to infer the best placement of the negative parts for any choice of (l_0, c) . Similarly, for a positive example, we use the stored model to infer the best placement of the positive parts for any choice of (l_0, c) . Once the latent variables are inferred, we compute the score of the complete latent configuration with respect to the *current model* to decide whether to add it to the cache (as a hard example), or to ignore it (as an easy example). Despite the added computational burden, this works reasonably fast in practice because we infer the value of the fixed latent variables only once per each data mining iteration. The optimization of the cache is done the same as in DPM.

The data mining step in GDPM is different from that of DPM in two ways. The first difference is in the way we add entries to the cache. Unlike DPM that stores only one entry for each positive training sample in the cache, in GDPM we may have multiple cache entries associated to a positive (as well as a negative) training sample. The other difference is that, unlike DPM that only has *hard negatives*, in GDPM we have *hard examples* that can be either positive or negative.

4.3.3 Initializing Negative Parts

Similar to DPM [20], we initialized the location and appearance of parts by training a detector without any parts; *i.e.* only a single root filter. We then select the sub-patch of the root filter with the highest positive/negative energy. The positive/negative energy of a sub-patch is measured by the sum of the squared values of the positive/negative weights in the sub-patch. The selected sub-patch is used to initialize a positive/negative part. We then set the values in the selected sub-patch of the root filter to zero and repeat the process to initialize the next positive/negative part.

We observed that negative parts in GDPM are particularly sensitive to initialization. This is due to the non-convexity of the optimization. Furthermore, while the positive label corresponds to a single visual class the negative label encompasses more classes and thus exhibits a multi-modal distribution.

Another approach to initialize negative parts is to use the positive parts from models of other objects. For example, *cow* is visually similar to *horse*, and analyzing the results of DPM also reveals that these two objects are often confused by DPM. Our experiments show that while a DPM *cow* detector with 8 positive parts obtains 21.9 average precision, adding one negative part to the model and initializing it with a part from the *horse* model improves the performance to 26.7 (see Figure 4-1).

4.4 Experimental Results

To demonstrate the efficacy of a GLVM we evaluate it on deformable part models (DPMs) by augmenting DPMs with negative parts. As discussed in Section 4.2 adding negative parts changes the optimization framework of DPMs. We conducted experiments on two different datasets. Similar to [20], we use HOG [14] as feature descriptor but the observed trends should be independent of this choice. We measure performance by the Average Precision of PR-curve. We use the PASCAL VOC criterion of 50% intersection over union for *recall*.

PASCAL VOC 2007 Animals. We first evaluate GDPM on the animal categories of VOC 2007, *i.e.* *bird*, *cat*, *cow*, *dog*, and *sheep*. The training set of VOC 2007

	bird	cat	cow	dog	horse	sheep
DPM₈	10.9	16.6	21.9	12.5	56.0	18.0
GDPM₇¹	10.9	17.0	22.0	12.5	57.2	19.2
GDPM₆²	9.5	18.6	23.2	12.2	56.0	19.8
GDPM₅³	10.5	13.9	21.7	11.9	54.8	19.1

Table 4.1: Average Precision (AP) for animal classes of PASCAL VOC 2007 dataset, comparing DPM with GDPM with different number of parts. Sub (super) indices indicate the number of positive (negative) parts.

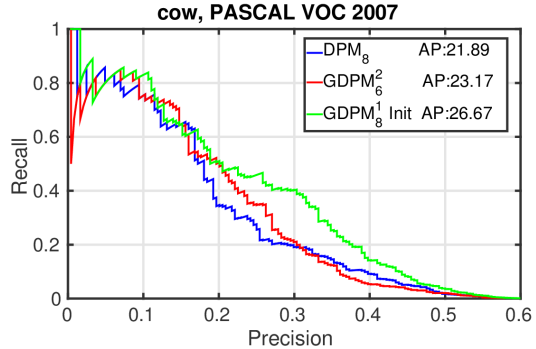


Figure 4-1: PR-curve for the *cow* class. We show result for DPM with 8 parts (blue), GDPM with 6 positive and 2 negative parts initialized (red), and GDPM with 8 positive and 1 negative part initialized by horse DPM model (green).

contains about 4500 negative images and a few hundreds (~ 400) positive instances per animal class. The same goes for the test set. We use the same hyper-parameters as used for original DPM [20], namely 6 components ($3 + 3$ mirror components) and 8 parts per component. Our results slightly differ from those reported in [20] due to the absence of post-processing (bounding box prediction and context re-scoring) and slight differences in implementation. Table 4.1 summarizes the results for substituting some positive parts in DPM with negative parts. 4 out of the 6 tested classes benefit from replacing positive parts with negative parts. Depending on the class the optimal number of negative parts is different. Also, Figure 4-1 shows the Precision Recall curve for the *cow* class. Although GDPM performs better than DPM in general, negative parts are particularly sensitive to initialization.

Cat Head. The second task is detection of heads of 12 breeds of cats in the Oxford Pets dataset [49]. For each breed of cats the distractor set contains images of the other 11 breeds and 25 breeds of dogs. Each breed of cat has around 65 positive images and 3500 negative images for training. Test set has the same number of images. This task is particularly hard due to the low number of positive training images and the high level of similarity between cats heads, making it a challenging tasks even for humans. Table 4.2 compares performance of DPM and GDPM using various number













												
	Abyssinian	Bengal	Birman	Bombay	British Shorthair	Egyptian Mau	Maine Coon	Persian	Ragdoll	Russian Blue	Siamese	Sphynx
DPM ₄	21.3	12.8	34.5	23.3	32.2	15.8	21.6	28.0	19.4	24.0	29.4	22.7
DPM ₆	22.2	12.8	31.4	21.5	31.3	16.5	26.0	29.0	20.6	25.0	30.9	22.0
GDPM ₂ ²	24.3	11.9	38.4	23.9	31.0	19.7	27.7	29.7	24.5	29.9	35.9	27.4
GDPM ₄ ²	25.5	13.7	34.0	23.0	33.5	20.9	24.5	30.0	21.9	25.3	30.6	23.2

Table 4.2: **Cat Head Detection:** Results comparing DPM and GDPM with different number of positive and negative parts. Sub (super) indices show the number of positive (negative) parts. GDPM consistently outperforms DPM variants. The average cat head images are taken from [76].

of positive/negative parts. Because of the scarcity of positive samples in the training set, we used one component per detector in this experiment. GDPM consistently outperforms DPM while using the same total number of parts. The header images in Table 4.2 are aligned average images generated by [76]. For a few classes (*e.g.* Bengal) DPM with 4 positive parts outperforms GDPM with 2 positive and 2 negative parts, but as we increase the number of parts to 6, replacing 2 positive parts with 2 negative parts proves beneficial. We observed that adding more parts (beyond 6) degrades the results. This can be due to over-fitting to the small number of positive images.

4.5 Conclusion

In this chapter we proposed a generalization of the family of latent variable models, called GLVM, that *explicitly* captures the *counter evidence* associated with the negative latent variables along with the evidence captured by the positive latent variables. We showed that GLVMs can model complex score functions that can be expressed as arbitrarily interleaved maximizations and minimizations of latent variables.

We also described a general framework for training GLVMs and showed how DPMs can benefit from the new perspective. Finally, we experimented on two object detection tasks revealing the benefits of negative latent variables. We further observed that expressive power of GLVM crucially depends on a careful initialization.

Chapter 5

Generalized

Majorization-Minimization

Non-convex optimization is ubiquitous in computer vision and machine learning. The Majorization-Minimization (MM) procedure systematically optimizes non-convex objectives through an iterative construction and optimization of upper bounds. The bound at each iteration is required to *touch* the objective function at the optimizer of the previous bound. We show that this touching constraint is unnecessary and overly restrictive. We generalize MM by relaxing this constraint, and propose a new optimization framework, named Generalized Majorization-Minimization (G-MM) that is more flexible compared to MM. For instance, it can incorporate application-specific biases into the optimization procedure without changing the objective function. We derive G-MM algorithms for several latent variable models and show that they consistently outperform their MM counterparts in optimizing non-convex objectives. In particular, G-MM algorithms appear to be less sensitive to initialization.

5.1 Background

Many problem in computer vision and machine learning entail non-convex For example, training object detectors from weakly labeled data [53, 61, 56], training classifiers with latent variables [3, 51, 20, 38], low-level vision tasks [10], and data clustering

[42, 1] all lead to non-convex optimization problems.

Majorization-Minimization (MM) [28] is a framework for designing algorithms for optimizing non-convex functions. MM iteratively optimizes a sequence of easy-to-optimize surrogate functions that bound the objective function, and guarantees convergence to a local extremum or a saddle point of the function. Two of the most successful instances of MM algorithms are Expectation Maximization (EM) [15] and Concave-Convex Procedure (CCCP) [72]. Both methods, however, have a number of drawbacks in practice, such as sensitivity to initialization and lack of uncertainty modeling for latent variables. This has been noted in the past in [43, 20, 47, 34, 50].

We propose a new procedure, *Generalized Majorization-Minimization (G-MM)*, for optimizing non-convex objective functions. Our approach is inspired by MM, but we generalize the bound construction process. Specifically, we relax the strict bound construction method in MM to allow for a *set* of valid bounds to be used, while still maintaining algorithmic convergence. This relaxation gives us more freedom in bound selection and can be used to design better optimization algorithms. For example, we can enforce certain constraints or encourage desired biases on the solution while leaving the original objective function intact. We incorporate this information when selecting the next bound from the set of valid bounds.

In training latent variable models and in clustering problems, MM algorithms such as CCCP and k -means are known to be sensitive to the initial values of the latent variables or cluster memberships. We refer to this problem as *stickiness* of the algorithm to the initial latent values. Our experimental results show that G-MM leads to methods that tend to be less sticky to initialization. We demonstrate the benefit of using G-MM on multiple problems, including k -means clustering and applications of Latent Structural SVMs to image classification with latent variables.

Both MM and G-MM are guaranteed to “make progress” in all iterations. A major difference, however, is in the way they measure progress. MM makes progress with respect to the value of the original objective function. To enforce this, MM requires the bound at each iteration to touch the objective function at the solution of the previous iteration. This “touching” requirement is very restrictive and, in practice,

can give rise to stickiness [47, 20]. It also overlooks the possibility of using bounds that do not touch the objective function but still make progress with respect to it.

In G-MM, bounds are constructed by leveraging prior beliefs about the model and/or data-driven constraints. This can be particularly useful when incorporating such *side-information* in the objective function directly is conceptually or computationally hard. Avoiding dead (or unbalanced) clusters in clustering and encouraging coherent solutions in image labeling are two examples of this (see Section 5.4 for more discussion). In cases where no prior belief is available to bias the selection of new bounds we propose to pick a bound *uniformly at random* from the family of valid bounds. The touching requirement in MM results in committing to a bound that agrees with the current solution better than all the other valid bounds. Thus, we expect G-MM with randomly selected bounds to be less sticky than MM in average.

5.1.1 Related Work

Perhaps the most famous iterative algorithm for non-convex optimization in machine learning and statistics is the EM algorithm [15]. EM is best understood in the context of maximum likelihood estimation in the presence of missing data, or *latent variables*. EM is a bound optimization algorithm: in each E-step, a lower bound on the likelihood is constructed, and the M-step maximizes this bound.

Countless efforts have been made to extend the EM algorithm since its introduction. In [43] it is shown that, while both steps in EM involve optimizing some functions, it is not necessary to fully optimize the functions in each step; in fact, each step only needs to “make progress”. This relaxation can potentially avoid bad local minima and even speed up convergence.

Lange *et al.* [28] proposed the Majorization-Minimization framework. MM generalizes methods like EM by “transferring” the optimization to a sequence of surrogate functions (bounds) on the original objective function. The Concave-Convex Procedure (CCCP) [72] is another widely-used instance of MM, where the surrogate function is obtained by *linearizing* the concave part of the objective function. Many successful machine learning algorithms employ CCCP, *e.g.* the Latent SVM [20].

Despite widespread success, MM (and CCCP in particular) has a number of drawbacks, some of which have motivated our work. CCCP is often sensitive to initialization, which necessitates expensive initialization or multiple trials [47, 11, 61]. In optimizing latent variable models, CCCP lacks the ability to incorporate application-specific information like latent variable uncertainty [34, 50] or posterior regularization [4] without making modifications to the objective function. Our framework is designed to deal with these drawbacks. Our key observation is that we can relax the constraint enforced by MM that requires the bounds to touch the objective function.

A closely related work to ours is pseudo-bound optimization by Tang *et al.* [64], which generalizes CCCP by using “pseudo-bounds” that may intersect the objective function. In contrast, our framework still uses valid bounds but only relaxes the touching requirement. Also, [64] is not as general as our framework in that it is designed specifically for optimizing binary energies in MRFs, and it restricts the form of surrogate functions to parametric max-flow.

The incremental and sparse variants of EM proposed in [43] are closely related to our work when we restrict our attention only to the EM algorithm. Typically, the likelihood function $L(\theta)$ is parameterized only by the model parameters θ . Neal *et al.* [43] formulate the likelihood as a function $F(\theta, q)$, where q is the posterior distributions of the latent variables, and show that if (θ, q) is a local maxima of F then θ is a local maxima of L as well. However, they do not provide an algorithm for how to optimize F . They only analyze a simple case where the function F is optimized in a 2-step iterative fashion, like EM. In the E-step, they update the posterior distributions of the latent variables only for a fraction of the training examples (instead of all of them). They show that, their algorithm converges faster than the standard EM because the E-steps are cheap to compute. This also fits in our framework. Besides that, our framework uses the exact same objective that is used in EM the only difference is in the bound selection step. We propose two strategies for selecting bounds (*stochastic* and *deterministic*) and demonstrate that they lead to higher quality solutions and are less sensitive to initialization than other EM-like methods.

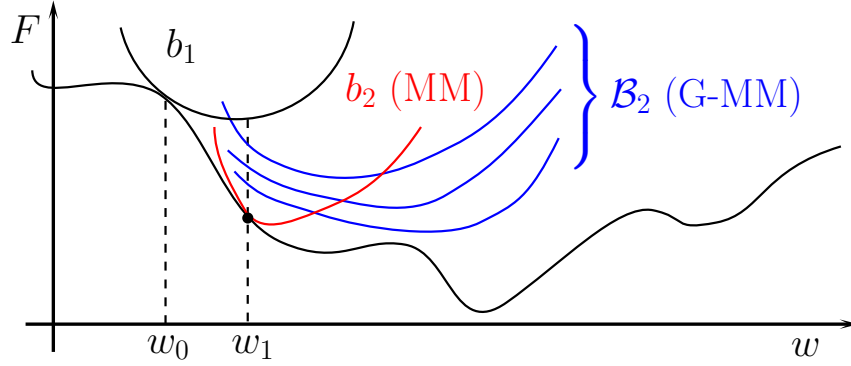


Figure 5-1: Optimization of a function F using MM (red) and G-MM (blue). In MM the bound b_2 has to touch F at w_1 . In G-MM we only require that b_2 be below b_1 at w_1 , leading to several choices \mathcal{B}_2 .

Algorithm 6 G-MM optimization

Input: w_0, η, ϵ

- 1: $v_0 := F(w_0)$
- 2: **for** $t := 1, 2, \dots$ **do**
- 3: select $b_t \in \mathcal{B}_t = \mathcal{B}(w_{t-1}, v_{t-1})$ as defined in (5.6)
- 4: $w_t := \arg \min_w b_t(w)$
- 5: $d_t := b_t(w_t) - F(w_t)$
- 6: $v_t := b_t(w_t) - \eta d_t$
- 7: **if** $d_t < \epsilon$ **break**
- 8: **end for**

Output: w_t

5.2 General Optimization Framework

We consider minimization of functions that are bounded from below. The extension to maximization is trivial. Let $F(w) : \mathbb{R}^d \rightarrow \mathbb{R}$ be the objective function that we wish to minimize. We propose an iterative procedure that generates a sequence of solutions w_1, w_2, \dots until it converges. The solution at iteration t is obtained by minimizing an upper bound $b_t(w)$ to the objective function *i.e.* $w_t = \arg \min_w b_t(w)$. The bound at iteration t is chosen from a set of “valid” bounds \mathcal{B}_t (see Figure 5-1). In practice, we assume that the members of \mathcal{B}_t come from a family \mathcal{F} of functions that upper bound F and can be optimized efficiently, such as quadratic functions, or quadratic functions with linear constraints. Algorithm 6 gives the outline of the approach. This general scheme is used in both MM and G-MM. However, as we shall see in the rest

of this section, MM and G-MM have key differences in the way they measure progress and the way they construct new bounds.

5.2.1 Progress Measure

MM measures progress with respect to the true objective values. In particular, MM requires that $\forall t : F(w_{t-1}) \geq F(w_t)$. To guarantee this MM requires that the bound at iteration t must touch the objective function at the previous solution, leading to the following constraint:

$$\text{MM constraint:} \quad b_t(w_{t-1}) = F(w_{t-1}). \quad (5.1)$$

This touching constraint, along with the fact that w_t minimizes b_t , and that b_t upper-bounds F , guarantees that MM improves the value of the true objective over time:

$$F(w_{t-1}) = b_t(w_{t-1}) \geq b_t(w_t) \geq F(w_t). \quad (5.2)$$

The touching requirement is restrictive and, in practice, can make the algorithm sensitive to initialization [47, 11, 61]. It also eliminates the possibility of using bounds that do not touch the objective function but may have other desirable properties.

In G-MM, we measure progress with respect to the bound values. It allows us to relax the touching constraint of MM, stated in (5.1), and require instead that,

$$\text{G-MM constraints:} \quad \begin{cases} b_1(w_0) = F(w_0) \\ b_t(w_{t-1}) \leq b_{t-1}(w_{t-1}). \end{cases} \quad (5.3)$$

Note that the G-MM constraints are weaker than MM. In fact, (5.1) implies (5.3) because b_{t-1} is an upper bound on F .

The weaker constraints used in G-MM do not imply $F(w_t) \leq F(w_{t-1})$, but, are sufficient to ensure that $\forall t, F(w_t) \leq F(w_0)$. This follows from the fact that b_t is an

upper bound of F , w_t is a minimizer of $b_t(w)$, and (5.3):

$$F(w_t) \leq b_t(w_t) \leq b_t(w_{t-1}) \leq \dots \leq b_1(w_1) \leq b_1(w_0) = F(w_0). \quad (5.4)$$

5.2.2 Bound Construction

This section describes step 3 of Algorithm 6. To construct new bounds, MM commits to a particular bound that touches the objective function F at the previous solution (see (5.3)). G-MM, however, considers a set of “valid” upper bounds that satisfy (5.3). Recall that \mathcal{F} is a family of functions that upper bound F and are “easy” to optimize; *e.g.* convex functions in \mathbb{R}^d :

$$\mathcal{F} = \{f : \mathbb{R}^d \rightarrow \mathbb{R} \mid f(w) \geq F(w), \forall w \in \mathbb{R} \text{ and } f \text{ is easy to optimize}\}. \quad (5.5)$$

We denote the set of valid bounds at iteration t by \mathcal{B}_t . To guarantee convergence, we restrict \mathcal{B}_t to bounds that are below a threshold v_{t-1} at the previous solution w_{t-1} :

$$\begin{aligned} \mathcal{B}_t &= \mathcal{B}(w_{t-1}, v_{t-1}) \\ \mathcal{B}(w, v) &= \{b \in \mathcal{F} \mid b(w) \leq v\}. \end{aligned} \quad (5.6)$$

Initially, we set $v_0 = F(w_0)$ to ensure that the first bound *touches* F , satisfying the first requirement of G-MM (see 5.3). In the next iterations, we set v_t using a *progress coefficient* $\eta \in (0, 1]$ that ensures making, at least, ηd_t progress where $d_t = b_{t-1}(w_{t-1}) - F(w_{t-1})$ is the gap between the previous bound and the true objective value at w_{t-1} . Note that when $\eta = 1$ all valid bounds touch F at w_{t-1} , corresponding to the MM requirement. Small η values allow for gradual exploratory progress in each step while large η values greedily select bounds that guarantee immediate progress.

We consider two scenarios for constructing bounds in G-MM and propose a deterministic and a stochastic approach for generating bounds as a result.

Deterministic Bound Generation: in the first scenario we assume we have access to a bias function $g : \mathcal{B}_t \times \mathbb{R}^d \rightarrow \mathbb{R}$ over the set of valid bounds. The function g

takes in a bound $b \in \mathcal{B}_t$ and a current solution $w \in \mathbb{R}^d$ and returns a scalar indicating the goodness of the bound. In this scenario we select the bound that has the largest bias value *i.e.* $b_t = \arg \max_{b \in \mathcal{B}_t} g(b, w_{t-1})$. MM algorithms, such as EM and CCCP, fall into this category.

Stochastic Bound Generation: in the second scenario we assume that all bounds are equally favorable and propose to choose one of the valid bounds from \mathcal{B}_t uniformly at random.

In general, MM algorithms are special cases of deterministic G-MMs that use a specific bias function $g(b, w) = -b(w)$. The bound that touches F at w maximizes this bias function. It also maximizes the amount of progress with respect to the previous bound at w . By choosing bounds that maximize progress, MM algorithms tend to rapidly converge to a nearby local minimum. For instance, at iteration t , the CCCP bound for latent SVMs is obtained by fixing latent variables in the concave part of the objective function to the best values according to the previous solution w_{t-1} , making the new solution $w_t = \arg \min_w b_t(w)$ attracted to w_{t-1} . Similarly, in the E-step, EM sets the posterior distribution of the latent variables conditioned on the data according to the model from the previous iteration. Thus, in the next maximization step, the model is updated to “match” the fixed posterior distributions. This explains one reason why MM algorithms are observed to be sensitive to initialization.

G-MM offers a more flexible bound construction scheme than MM. The bias function g can incorporate prior beliefs about the desired properties of the selected bounds whenever such prior information is available. Importantly, unlike in MM, G-MM can use any bias function. In Section 5.5 we show empirically that picking bounds uniformly at random from the set of valid bounds is less sensitive to initialization and leads to a better result compared to CCCP and k -means (hard-EM). We also show that using good bias functions can further improve performance of the learned models.

5.3 Convergence Proof of G-MM

We showed in (5.4) that, in any iteration t , the G-MM solution is no worse than the initial solution in terms of the objective value, *i.e.* $F(w_t) \leq F(w_0)$. In this section we also show that the gap between the G-MM bounds and the true objective converges to zero at the minimizers of the bounds (Theorem 5.1). Moreover, we show, under mild conditions, G-MM converges to a solution (Theorem 5.2) that is a local extremum or a saddle point of F (Theorem 5.3). Other convergence properties of G-MM depend on the structure of the objective function F , the family of the bounds \mathcal{F} , and the details of the bound selection strategy.

Theorem 5.1. *In the limit as $t \rightarrow \infty$, G-MM bounds and F touch at the minimizer of the bounds, i.e. $\lim_{t \rightarrow \infty} d_t = 0$.*

Proof. We have $b_t(w_t) \leq b_t(w_{t-1}) \leq v_{t-1}$. The first inequality holds because w_t minimizes b_t and the second inequality follows from (5.6). Summing over t gives

$$\sum_{t=1}^T b_t(w_t) \leq \sum_{t=1}^T v_{t-1} = v_0 + \sum_{t=1}^{T-1} b_t(w_t) - \eta d_t \quad (5.7)$$

$$\Rightarrow \eta \sum_{t=1}^T d_t \leq v_0 - b_T(w_T) = F(w_0) - b_T(w_T), \quad (5.8)$$

where $\forall t \geq 1$, v_t is defined as in line 6 of Algorithm 6. Let $w^* \in \arg \min_w F(w)$ be an unknown *global* minimum of F . Using $F(w^*) \leq F(w_T) \leq b_T(w_T)$ and (5.8), we have

$$\eta \sum_{t=1}^T d_t \leq F(w_0) - F(w^*) \Rightarrow \lim_{t \rightarrow \infty} d_t = 0. \quad (5.9)$$

Since we assume F is bounded from below, if F does not have a unique global minimum we can replace $F(w^*)$ with the value of lower bound in (5.9). \square

Theorem 5.2. *If $\forall b \in \mathcal{F}$, b is differentiable and $\nabla^2 b(w) \succeq mI$ for $m > 0$, where I is the identity matrix, then G-MM converges to a solution; i.e. $\exists w^\dagger$ s.t. $\lim_{t \rightarrow \infty} w_t = w^\dagger$.*

Proof. If $\nabla^2 f(x) \succeq mI$, for any two points x, y in the domain of f we have:

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + (m/2)\|x - y\|^2 \quad (5.10)$$

We set $f = b_t$, $x = w_t$, and $y = w_{t-1}$ in (5.10), and note that $\nabla b_t(w_t) = 0$ to get:

$$(2/m)(b_t(w_{t-1}) - b_t(w_t)) \geq \|w_t - w_{t-1}\|^2 \quad (5.11)$$

We can replace $b_t(w_{t-1})$ by v_{t-1} in (5.11) due to (5.6). By adding the inequalities for T iterations, similar to what we did in the proof of Theorem 5.1, we get:

$$(2/m)(F(w_0) - F(w^*)) \geq \sum_{t=1}^T \|w_t - w_{t-1}\|^2 \quad (5.12)$$

$$\Rightarrow \exists w^\dagger \text{ s.t. } \lim_{t \rightarrow \infty} w_t = w^\dagger \quad (5.13)$$

□

Theorem 5.3. *If F is differentiable and $\forall b \in \mathcal{F}$, b is differentiable and $MI \succeq \nabla^2 b(w) \succeq mI$ for $\infty > M > m > 0$, where I is the identity matrix, then $\nabla F(w^\dagger) = 0$; that is, G -MM converges to a local extremum or saddle point of F .*

Proof. We briefly overview the sketch of the proof before explaining the steps in detail. We prove the theorem by contradiction and show that if $\nabla F(w^\dagger) \neq 0$ then we can construct a lower bound on F (stated in (5.21)) that, at some point w , goes above an upper bound of b_t (stated in (5.25)). This implies that $\exists w$ s.t. $F(w) > b_t(w)$ which contradicts with the assumption that b_t is an upper bound of F . The proof has three steps: 1) constructing the lower bound on F , 2) constructing the upper bound on b_t , and 3) finding a point w s.t. $F(w) > b_t(w)$. Details of the proof are presented below.

Step 1: Assume $\|\nabla F(w^\dagger)\|_2 \neq 0$. This can be stated as

$$\exists A > 0 \text{ s.t. } \|\nabla F(w^\dagger)\|_2 = 2A. \quad (5.14)$$

Consider value of F on the line that touches F at w^\dagger . We denote this by $g(z)$ where

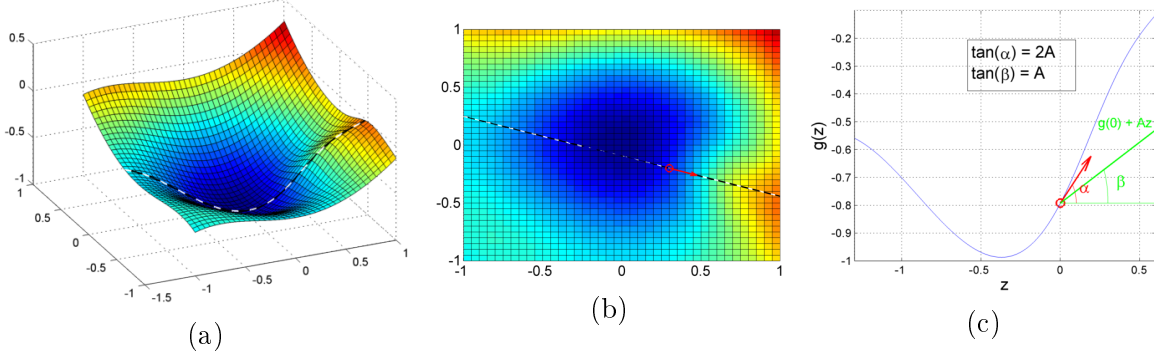


Figure 5-2: An example of a non-convex function F on \mathbb{R}^d . (a) shows the plot of F ; the value of the function on the line that touches F at the point $p = (0.3, -0.2)$ is marked by the black and white line. (b) shows F from the top view; it also marks the point p with the red circle and shows the direction of the gradient vector at p . The length of the vector is set arbitrarily for visualization purposes. (c) shows the value of F along the tangent line as a one dimensional function (see Equation 5.15); it also shows the gradient direction at $z = 0$ (in red) and the linear function of Equation 5.21 that lower bounds F (in green).

$z \in \mathbb{R}$ and define it as

$$g(z) = F(w^\dagger + zu), \quad u = \frac{\nabla F(w^\dagger)}{\|\nabla F(w^\dagger)\|_2}. \quad (5.15)$$

We can write the following equalities:

$$g'(z) = \frac{\partial g}{\partial z}(z) = \nabla F(w^\dagger + zu) \cdot u \quad (5.16)$$

$$g'(0) = \|\nabla F(w^\dagger)\|_2 = 2A \quad (5.17)$$

$$g(0) = F(w^\dagger) \quad (5.18)$$

In Figure 5-2a we show a non-convex function F in \mathbb{R}^2 . The line that touches the function at a particular point p is also shown in the figure. Figure 5-2b shows the same function from the top view, together with the gradient direction at p . Figure 5-2c shows how the value of F changes along the tangent line as a function of the step size z along the gradient direction u (see $g(z)$ in Equation 5.15).

Since F is differentiable and its gradient changes smoothly, so is g , and we can

use the mean value theorem to rewrite g for $z \geq 0$ as follows ¹:

$$\forall z \geq 0, \exists k \in [0, z] \text{ s.t. } g(z) = g(0) + g'(k)z. \quad (5.19)$$

Smoothness of g also implies that in a small neighborhood around 0 the gradient of g is larger than A ; that is:

$$\exists \delta > 0 \text{ s.t. } \forall z \in [0, \delta], g'(z) > A. \quad (5.20)$$

We use this inequality in (5.19) to get a lower bound on g in vicinity of 0, or equivalently, a lower bound on F in the vicinity of w^\dagger and in the direction of u , as follows:

$$\begin{aligned} \forall z \in [0, \delta], g(z) &\geq g(0) + Az \Rightarrow \\ \forall z \in [0, \delta], F(w^\dagger + zu) &\geq F(w^\dagger) + Az. \end{aligned} \quad (5.21)$$

This lower bound is shown in Figure 5-2c in green.

Step 2: To get an upper bound on b_t we note that $w_t = \arg \min_w b_t(w)$ is the minimizer of b_t and since b_t is differentiable, $\nabla b_t(w_t) = 0$. From this and the assumption that the curvature of all bounds in all directions is finite (*i.e.* $\forall b \in \mathcal{F}, \nabla^2 b(w) \preceq MI$) we can upper bound b_t with a quadratic function with curvature M whose minimum occurs at $w = w_t$ and has value $b_t(w_t)$, as follows:

$$\forall w, (w - w_t)^T MI (w - w_t) + b_t(w_t) \geq b_t(w). \quad (5.22)$$

Similar to step 1, we only focus the points w that lie on the line $w^\dagger + zu$ and get the following bound on b_t for all $z \in \mathbb{R}$:

$$M \|w^\dagger + zu - w_t\|^2 + b_t(w_t) \geq b_t(w^\dagger + zu) \quad (5.23)$$

From Young's inequality in multiple dimensions we have that $\forall a, b \in \mathbb{R}^d, \|a + b\|^2 \leq$

¹ g can be written similarly for $z < 0$, however, we are only interested in the positive half space.

$2\|a\|^2 + 2\|b\|^2$. We can set $a = w^\dagger - w_t$ and $b = zu$ and use Young's inequality in the left hand side of (5.23) to get:

$$2M\|w^\dagger - w_t\|^2 + 2Mz^2 + b_t(w_t) \geq b_t(w^\dagger + zu) \quad (5.24)$$

From Theorem 5.2 we have that $\forall \epsilon_1 > 0, \exists T_1$ s.t. $\forall t > T_1, \|w^\dagger - w_t\|_2 < \epsilon_1$. We can use this in (5.24) to get the following upper bound for b_t :

$$2M\epsilon_1^2 + 2Mz^2 + b_t(w_t) \geq b_t(w^\dagger + zu) \quad (5.25)$$

Step 3: We show that $\exists z \in [0, \delta]$ for which the right hand side of (5.21) is larger than the left hand side of (5.25), which implies that for $w = w^\dagger + zu, F(w) > b_t(w)$.

We need to find $z \in [0, \delta]$ such that

$$\begin{aligned} F(w^\dagger) + Az &> 2M\epsilon_1^2 + 2Mz^2 + b_t(w_t) \equiv \\ 2Mz^2 - Az + 2M\epsilon_1^2 + b_t(w_t) - F(w^\dagger) &< 0. \end{aligned} \quad (5.26)$$

Lemma 5.1. $\forall \epsilon_2 > 0, \exists T_2$ s.t. $\forall t > T_2, b_t(w_t) - F(w^\dagger) < \epsilon_2$.

Proof. To prove the lemma we first show that

$$\lim_{t \rightarrow \infty} |b_t(w_t) - F(w^\dagger)| = 0 \quad (5.27)$$

and then note that $b_1(w_1), b_2(w_2), \dots$ is a non-increasing sequence, and therefore, $\forall w, b_t(w_t) \geq F(w^\dagger)$, thus, we can drop the absolute value.

To show (5.27), we note that we can write:

$$\begin{aligned} |b_t(w_t) - F(w^\dagger)| &= |(b_t(w_t) - F(w_t)) + (F(w_t) - F(w^\dagger))| \\ &\leq |b_t(w_t) - F(w_t)| + |F(w_t) - F(w^\dagger)|. \end{aligned}$$

The first term on the right hand side can get arbitrarily close to zero due to Theorem 5.1. The second term, also, can get arbitrarily close to zero due to Theorem 5.2 and

that F it is Lipschitz continuous (because it is differentiable). \square

We use lemma 5.1 in inequality (5.26) to get

$$2Mz^2 - Az + 2M\epsilon_1^2 + \epsilon_2 < 0. \quad (5.28)$$

Note that ϵ_1 and ϵ_2 can be chosen arbitrarily. In particular, we can set them so that the discriminant of the quadratic function in the left hand side of (5.28) is positive, as follows:

$$\begin{aligned} \Delta &= A^2 - 4(2M)(2M\epsilon_1^2 + \epsilon_2) > 0 \\ \Rightarrow 2M\epsilon_1^2 + \epsilon_2 &< \frac{A^2}{8M}. \end{aligned} \quad (5.29)$$

This guarantees that the quadratic function in (5.28) has two *distinct* roots. If $2M\epsilon_1^2 + \epsilon_2 = 0$, the two roots are $z_1 = 0$ and $z_2 = -\frac{b}{a} = \frac{A}{2M} > 0$. Changing ϵ_1 and ϵ_2 only affects the constant $2M\epsilon_1^2 + \epsilon_2$, and thereby, it only shifts the quadratic function up or down. In particular, we can control ϵ_1 and ϵ_2 to drive the constant arbitrarily close to 0, causing $z_1 > 0$ to get arbitrarily close to zero. In particular, since $\delta > 0$, we can set ϵ_1 and ϵ_2 such that $0 < z_1 < \frac{\delta}{2}$, and also (5.29) holds. Then $\forall z \in (z_1, \min(z_2, \delta))$ inequality (5.28) holds and we have that $F(w^\dagger + zu) > b_t(w^\dagger + zu)$. This violates the assumption that b_t is an upper bound of F and proves that our initial hypothesis (stated in (5.14)) is wrong and, therefore, $\|\nabla F(w^\dagger)\|_2 = 0$.

\square

5.4 Examples of Derived G-MM Algorithms

In this section we derive G-MM algorithms for k -means clustering and Latent Structural SVM (LS-SVM), both of which are widely used in machine learning. To this end, we implement the bound construction step (line 3) and the optimization step (line 4) of Algorithm 6. We primarily focus on demonstrating G-MM on *latent variable models* where bound construction naturally corresponds to imputing latent variables in the

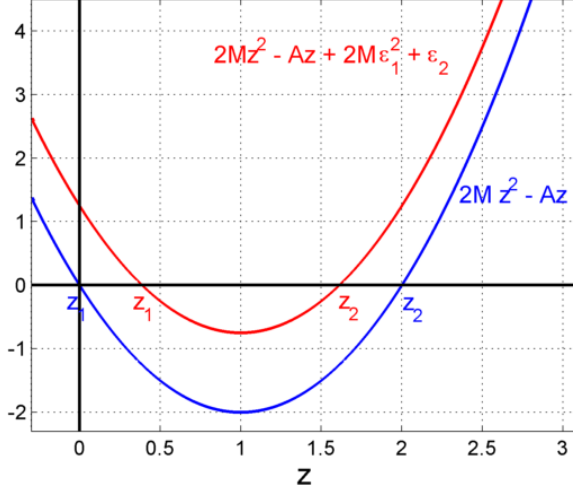


Figure 5-3: The quadratic function in the left hand side of (5.28). When $\epsilon_1 = \epsilon_2 = 0$, the quadratic function has two roots, $z_1 = 0$ and $z_2 = \frac{A}{2M} > 0$ (shown in blue). When ϵ_1 and ϵ_2 are non-zero, but chosen so that (5.29) holds, the quadratic function is shifted up but still has two distinct roots (shown in red).

model. Both k -means and LS-SVM belong to this category. It is worth mentioning that G-MM is fully capable of handling more general non-convex problems.

5.4.1 Clustering

Let $\{x_1, \dots, x_n\}$ denote a set of sample points and $w = (\mu_1, \dots, \mu_k)$ denote a set of cluster centers. We use $z_i \in \{1, \dots, k\}$ to denote the index of the cluster assigned to the i -th sample point. The objective function in k -means is defined as follows:

$$F(w) = \sum_{i=1}^n \min_{z_i=1}^k \|x_i - \mu_{z_i}\|^2. \quad (5.30)$$

Bound construction: We obtain a convex upper bound on F by fixing the latent variables (z_1, \dots, z_n) to certain values instead of minimizing over these variables. Bounds constructed this way are quadratic convex functions of $w = (\mu_1, \dots, \mu_k)$,

$$\mathcal{F} = \left\{ b(w) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2 \mid \forall i, z_i \in \{1, \dots, k\}^n, \quad w = (\mu_1; \dots; \mu_k) \right\}. \quad (5.31)$$

The k -means algorithm is an instance of MM. The algorithm repeatedly assigns each example to its nearest center to construct a bound, and then updates the centers by optimizing the bound. We can set $g(b, w) = -b(w)$ in G-MM to obtain the k -means algorithm. We can also define g differently to obtain a G-MM algorithm

that exhibits other desired properties. For instance, a common issue in clustering is cluster starvation. One can design a bias function that encourages balanced clusters by selecting g appropriately.

We select a random bound from \mathcal{B}_t by sampling a latent configuration $z = (z_1, \dots, z_n)$ uniformly from the set of configurations leading to valid bounds. Specifically, we start from a valid initial configuration (*e.g.* k -means solution) and perform a random walk on a graph whose nodes are latent configurations defining valid bounds. The neighbors of a latent configuration z are the valid configurations that can be obtained by changing the value of one latent variable. The transition probabilities are chosen properly to define a uniform stationary distribution on the graph.

Bound optimization: Optimization of a bound $b \in \mathcal{F}$ can be done in closed form by setting μ_j to be the mean of all examples assigned to cluster j :

$$\mu_j = \frac{\sum_{i \in I_j} x_i}{|I_j|}, \quad I_j = \{1 \leq i \leq n \mid z_i = j\}. \quad (5.32)$$

5.4.2 Detection and Classification with Latent Structural SVM

A Latent Structural SVM (LS-SVM) [71] defines a structured output classifier with latent variables. It extends the Structural SVM [30] by introducing latent variables.

Let $\{(x_i, y_i)\}_{i=1}^n$ denote a set of labeled examples with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We assume that each example x_i has an associated latent value $z_i \in \mathcal{Z}$. Let $\phi(x, y, z) : \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \rightarrow \mathbb{R}^d$ denote a *feature map*. A vector $w \in \mathbb{R}^d$ defines a classifier $\hat{y} : \mathcal{X} \rightarrow \mathcal{Y}$,

$$\hat{y}(x) = \arg \max_y (\max_z w \cdot \phi(x, y, z)). \quad (5.33)$$

The LS-SVM training objective is defined as follows,

$$F(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \left(\max_{y,z} (w \cdot \phi(x_i, y, z) + \Delta(y, y_i)) - \max_z w \cdot \phi(x_i, y_i, z) \right), \quad (5.34)$$

where λ is a hyper-parameter that controls regularization and $\Delta(y, y_i)$ is a non-

negative loss function that penalizes the prediction y when the ground truth label is y_i .

Bound construction: As in the case of k -means clustering a convex upper bound on the LS-SVM objective can be obtained by imputing latent variables. Specifically, for each example x_i , we fix $z_i \in \mathcal{Z}$, and replace the maximization in the last term of the objective with a linear function $w \cdot \phi(x_i, y_i, z_i)$. This forms a family of convex piecewise quadratic bounds,

$$\mathcal{F} = \left\{ \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \max_{y,z} (w \cdot \phi(x_i, y, z) + \Delta(y, y_i)) - w \cdot \phi(x_i, y_i, z_i) \mid \forall i, z_i \in \mathcal{Z} \right\}. \quad (5.35)$$

The CCCP algorithm for LS-SVM selects the bound b_t defined by $z_i^t = \arg \max_{z_i} w_{t-1} \cdot \phi(x_i, y_i, z_i)$. This particular choice is a special case of G-MM when $g(b, w) = -b(w)$.

To generate random bounds from \mathcal{B}_t we use the same approach as in the case of k -means clustering. We perform a random walk in a graph where the nodes are latent configurations leading to valid bounds, and the edges connect latent configurations that differ in a single latent variable.

Bound optimization: Optimization of a bound $b \in \mathcal{F}$ corresponds to a convex quadratic program and can be solved using different techniques, including gradient based methods (*e.g.* SGD) and the cutting-plane method [30]. We use the cutting-plane method in our experiments.

Bias Function for Multi-fold Multiple Instance Learning

The multi-fold MIL algorithm [11] was introduced for training latent SVMs for weakly supervised object localization, to deal with stickiness issues in training with CCCP. It modifies how latent variables are updated during training. [11] divides the training set into K folds, and updates the latent variables in each fold using a model trained on the other $K - 1$ folds. This algorithm does not have a formal convergence guarantee. By defining a suitable bias function, we can derive a G-MM algorithm that mimics

the behavior of multi-fold MIL, and yet, is convergent.

Consider training an LS-SVM. Let $S = (1, \dots, n)$ be an ordered sequence of training sample indices. Also, let $z_i \in \mathcal{Z}$ denote the latent variable associated to training example (x_i, y_i) . Let I denote a subsequence of S . Also, let z_I^t denote the fixed latent variable values of training examples in I in iteration t , and let $w(I, z_I^t)$ be the model trained on $\{(x_i, y_i) | i \in I\}$ with latent variables fixed to z_I^t in the last maximization of Equation 5.34.

We assume access to a loss function $\ell(w, x, y, z)$. For example, for latent SVM [20] where $y \in \{-1, 1\}$, ℓ is the hinge loss: $\ell(w, x, y, z) = \max\{0, 1 - yw \cdot \phi(x, z)\}$. We start by considering the Leave-One-Out (LOO) setting, *i.e.* $K = n$. We call [11]’s algorithm LOO-MIL in this case. The update rule of LOO-MIL in iteration t is:

$$z_i^t = \arg \min_{z \in \mathcal{Z}} \ell \left(w(S \setminus i, z_{S \setminus i}^{t-1}), x_i, y_i, z \right), \quad \forall i \in S. \quad (5.36)$$

After updating the latent values for all training examples, the model w is retrained by optimizing the resulting bound.

Now let us derive the bias function for a G-MM algorithm that mimics the behavior of LOO-MIL. Recall from Equation 5.35 that each bound $b \in \mathcal{B}_t$ is associated with a joint latent configuration $z(b) = (z_1, \dots, z_n)$. We use the following bias function:

$$g(b, w) = - \sum_{i \in S} \ell \left(w(S \setminus i, z_{S \setminus i}^{t-1}), x_i, y_i, z_i \right). \quad (5.37)$$

Note that picking a bound according to (5.37) is equivalent to the LOO-MIL update rule of (5.36) except that in (5.37) only *valid* bounds are considered; that is bounds that make at least η -progress.

For the general multi-fold case (*i.e.* $K < n$), the function g can be derived similarly.

5.5 Experimental Results

We evaluate G-MM and MM algorithms on clustering and LS-SVM training on various datasets. Recall from (5.6) that the progress coefficient η defines the set of valid

bounds \mathcal{B}_t in each step. CCCP and k -means bounds correspond to setting $\eta = 1$, thus taking maximally large steps towards a local minimum of the true objective.

5.5.1 Clustering with G-MMs

We conduct experiments on five different clustering datasets: Aggregation [23], Norm-25 [1], D31 [66], Cloud [1], and GMM-200. Norm-25, D31, and GMM-200 are synthetic and Cloud is from real data. See the references for details about the datasets. GMM-200 was created by us. It is a Gaussian mixture model on 2-D data with 200 mixture components. Each component is a Gaussian distribution with $\sigma = 1.0$ and μ on a square of size 70×70 . The means are placed at least 2.5σ apart from each other. The dataset contains 50 samples per mixture component.

We compare results from three different initializations: **forgy** selects k training examples uniformly at random without replacement to define initial cluster centers, **random partition** assigns training samples to cluster centers randomly, and **k-means++** uses the algorithm in [1]. In each experiment we run the algorithm 50 times and report the mean, standard deviation, and the best objective value (Equation 5.30). Table 5.1 shows the results using k -means (hard-EM) and G-MM. In each run, the two optimization methods are initialized from exactly the same point by setting the random seed to the same number. We note that the variance of the solutions found by G-MM is typically smaller than that of k -means. Moreover, the best and the average solutions found by G-MM are always better than (or the same as) those found by k -means. This trend generalizes over different initialization schemes as well as different datasets and suggests that the G-MM algorithm requires fewer runs to find a good solution.

Although *random partition* seems to be a very bad initialization for k -means on all datasets, G-MM recovers from it. In fact, on D31 and GMM-200 datasets, G-MM initialized by *random partition* performs better than when it is initialized by other methods (including k -means++). Also, the variance of the best solutions (across different initialization methods) in G-MM is smaller than that of k -means. These suggest that the G-MM optimization is less sticky to initialization than k -means.

Dataset	k	Opt. Method	forgy		random partition		k-means++	
			avg \pm std	best	avg \pm std	best	avg \pm std	best
Aggregation	7	hard-EM	14.93 \pm 1.15	13.96	55.81 \pm 61.52	14.47	14.81 \pm 1.04	13.96
		G-MM	14.34 \pm 0.97	13.96	14.31 \pm 1.08	13.96	14.43 \pm 0.84	13.96
Norm-25	25	hard-EM	$1.9 \times 10^5 \pm 2 \times 10^5$	70000	$5.8 \times 10^5 \pm 3 \times 10^5$	220000	$5.3 \times 10^3 \pm 9 \times 10^3$	1.5
		G-MM	$9.7 \times 10^3 \pm 1 \times 10^4$	1.5	$2.0 \times 10^4 \pm 0$	20000	$4.5 \times 10^3 \pm 8 \times 10^3$	1.5
D31	31	hard-EM	1.69 \pm 0.03	1.21	52.61 \pm 47.06	4.00	1.55 \pm 0.17	1.10
		G-MM	1.43 \pm 0.15	1.10	1.21 \pm 0.05	1.10	1.45 \pm 0.14	1.10
Cloud	50	hard-EM	1929 \pm 429	1293	44453 \pm 88341	3026	1237 \pm 92	1117
		G-MM	1465 \pm 43	1246	1470 \pm 8	1444	1162 \pm 95	1067
GMM-200	200	hard-EM	2.25 \pm 0.10	2.07	11.20 \pm 0.63	9.77	2.12 \pm 0.07	1.99
		G-MM	2.04 \pm 0.09	1.90	1.85 \pm 0.02	1.80	1.98 \pm 0.06	1.89

Table 5.1: Comparison of G-MM and k -means (hard-EM) on multiple clustering datasets. Three different initialization methods were compared; **forgy** initializes cluster centers to random examples, **random partition** assigns each data point to a random cluster center, and **k-means++** implements the algorithm from [1]. The mean, standard deviation, and best objective values out of 50 random trials are reported. Both k -means and G-MM use the exact same initialization in each trial. G-MM consistently converges to better solutions.

Figure 5-4 shows the effect of the progress coefficient on the quality of the solution found by G-MM. Different colors correspond to different initialization schemes. The solid line indicates the average objective over 50 iterations, the shaded area covers one standard deviation from the average, and the dashed line indicates the best solution over the 50 trials. Smaller progress coefficients allow for more extensive exploration, and hence, smaller variance in the quality of the solutions. On the other hand, when the progress coefficient is large G-MM is more sensitive to initialization (*i.e.* is more sticky) and, thus, the quality of the solutions over multiple runs is more diverse. However, despite the greater diversity, the best solution is worse when the progress coefficient is large. G-MM reduces to k -means if we set the progress coefficient to 1 (*i.e.* the largest possible value).

Figure 5-5 visualizes the result of k -means and G-MM on the D-31 dataset from the same initialization. G-MM finds a near perfect solution while, in k -means, many clusters get merged incorrectly or die off. Dead clusters are those which do not get any points assigned to them. The update rule of Equation 5.32 collapses the dead clusters on to the origin.

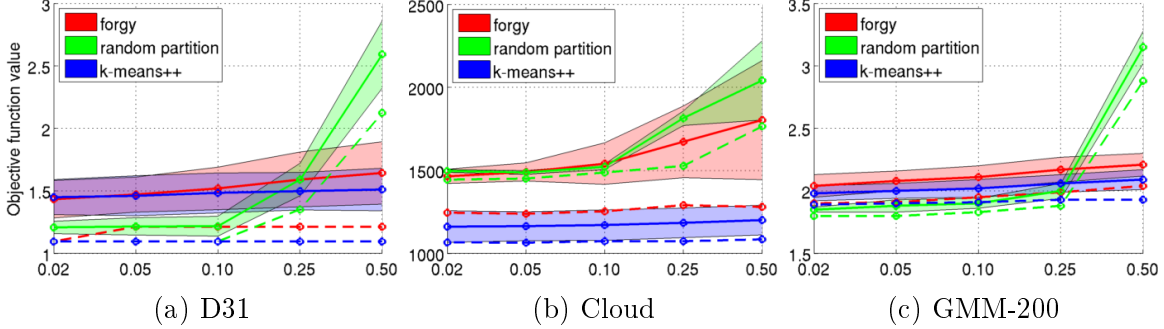


Figure 5-4: The effect of the progress coefficient η (x-axis) on the quality of the solutions found by G-MM on three different clustering datasets. The quality of the solutions is measured by the objective function in (5.30). Lower values are better. The average (solid line), the best (dashed line), and the variance (shaded area) over 50 trials are shown and different initializations are coded with different colors.

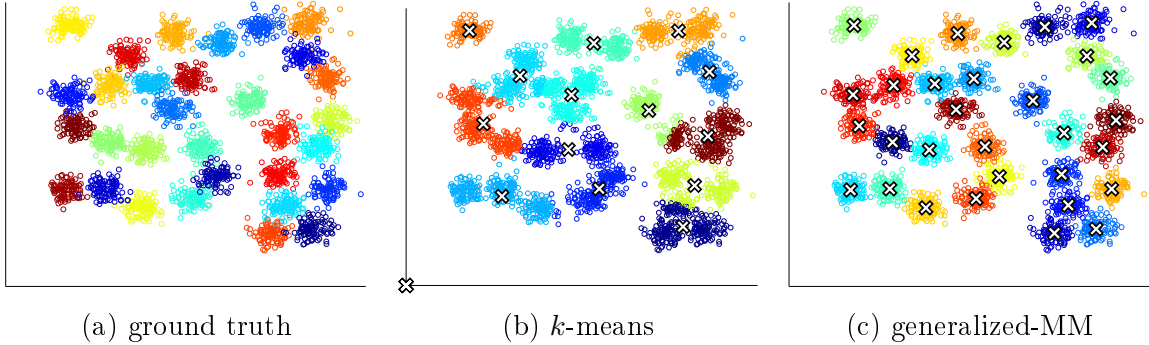


Figure 5-5: Visualization of the solution of k -means and G-MM on the D31 dataset [66] from identical starting point. Random partition initialization scheme is used. (a) color-coded ground-truth clusters. (b) solution of k -means. (c) solution of G-MM. The white crosses indicate location of the cluster centers. Color codes match up to a permutation.

5.5.2 Object Detection with G-MMs

We consider the problem of training an LS-SVM classifier on the mammals dataset [27]. The dataset contains images of six mammal categories with image-level annotation. Locations of the objects in these images are not provided, and therefore, treated as latent variables in the model. Specifically, let x be an image and y be a class label ($y \in \{1, \dots, 6\}$ in this case), and let z be the latent location of the object in the image. We define $\phi(x, y, z)$ to be a feature function with 6 blocks; one block for each category. It extracts features from location z of image x and places them in the y -th block of the output and fills the rest with zero. We use the following multi-class

Opt. Method	center		top-left		random	
	objective	test error	objective	test error	objective	test error
CCCP	1.21 ± 0.03	22.9 ± 9.7	1.35 ± 0.03	42.5 ± 4.6	1.47 ± 0.03	31.8 ± 2.6
G-MM random	0.79 ± 0.03	17.5 ± 3.9	0.91 ± 0.02	31.4 ± 10.1	0.85 ± 0.03	19.6 ± 9.2
G-MM biased	0.64 ± 0.02	16.8 ± 3.2	0.70 ± 0.02	18.9 ± 5.0	0.65 ± 0.02	14.6 ± 5.4

Table 5.2: LS-SVM results on the mammals dataset [27]. We report the mean and standard deviation of the training objective (Equation 5.34) and test error over five folds. Three strategies for initializing latent object locations are tried: image center, top-left corner, and random location. “G-MM random” uses random bounds, and “G-MM bias” uses a bias function inspired by multi-fold MIL [11]. Both variants consistently and significantly outperform the CCCP baseline.

classification rule:

$$y(x) = \arg \max_{y,z} w \cdot \phi(x, y, z), \quad w = (w_1, \dots, w_6). \quad (5.38)$$

Our experimental setup is similar to that in [34]. We use Histogram of Oriented Gradients [14] for the image feature ϕ , and the 0-1 classification loss for Δ . We set $\lambda = 0.4$ in (5.34), and report 5-fold cross-validation performance. We compare three strategies for initializing the latent object locations: *center* of the image, *top-left* corner, and *random* locations. The first is a reasonable initialization since most objects are at the center in this dataset; the second initialization strategy is adversarial.

We try a stochastic as well as a deterministic bound construction method. In each iteration t , the stochastic method samples a subset of examples S_t from the training set uniformly, and updates their latent variables using $z_i^t = \arg \max_{z_i} w_{t-1} \cdot \phi(x_i, y_i, z_i)$. Other latent variables are kept the same as the previous iteration. We increase the size of S_t across iterations by 20% of the size the entire training set across iterations.

The deterministic method uses the bias function described in Section 5.4.2. This is inspired by the multi-fold MIL idea of [11], and is shown to be robust to initialization, especially in high dimensions. We set the number of folds to 10 in our experiments. Table 5.2 shows results on the mammals dataset. Both variants of G-MM consistently outperform CCCP in terms of training objective and test error. We observed that CCCP rarely updates the latent locations, under all initializations. On the other hand, both variants of G-MM significantly alter the latent locations, thereby avoiding

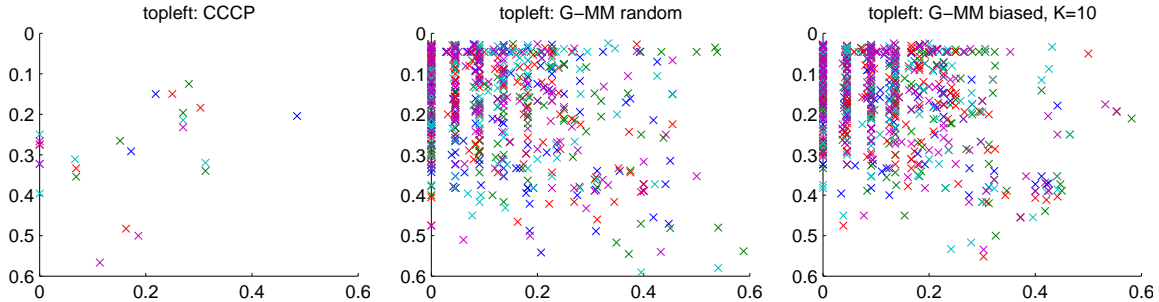


Figure 5-6: Latent location changes after learning, in relative image coordinates, for all five cross-validation folds, for the *top-left* initialization on the mammals dataset. Left to right: CCCP, “G-MM random”, “G-MM biased” ($K = 10$). Each cross represents a training image; cross-validation folds are color coded differently. Averaged over five folds, CCCP only alters 2.4% of all latent locations, leading to very bad performance. “G-MM random” and “G-MM biased” alter 86.2% and 93.6% on average, respectively, and perform much better.

the local minima close to the initialization. Figure 5-6 visualizes this for *top-left* initialization. Since objects rarely occur at the top-left corner in the mammals dataset, a good model is expected to significantly update the latent locations. Averaged over five cross-validation folds, about 90% of the latent variables were updated in G-MM after training whereas this measure was 2.4% for CCCP. This is consistent with the better training objectives and test errors of G-MM. Figure 5-7 shows example training images and the final imputed latent object locations by three algorithms: CCCP (red), G-MM random (blue), and G-MM biased (green). The initialization is *top-left*.

In most cases CCCP fails to update the latent locations given by initialization. The two G-MM variants, however, update the latent locations significantly and often localize objects in the training images correctly. This is achieved *only* with image-level object category annotations, and with a very bad (even adversarial) initialization.

5.5.3 Image Classification with G-MMs

We implement the reconfigurable model of Chapter 2 to do scene classification on MIT-Indoor dataset [52], which has images from 67 indoor scene categories. We segment each image into a 10×10 regular grid and treat the grid cells as image regions. We train a model with 200 shared parts. All parts can be used to describe the data



Figure 5-7: Example training images from the mammals dataset, shown with final imputed latent object locations by three algorithms: CCCP (red), G-MM random (blue), G-MM biased (green). Initialization: *top-left*.

in an image region. We use the pre-trained hybrid ConvNet of [73] to extract features from image regions. We record from the 4096 neurons at the penultimate layer of the network and use PCA to reduce the dimensionality of these features to 240.

The reconfigurable model is an instance of LS-SVMs where the latent variables are the assignments of parts to image regions and the output structure is the multi-valued category labels. We refer to Chapter 2 for more details about the model.

Initializing training entails the assignment of parts to image regions *i.e.* setting z_i 's in Equation 5.35 to define the first bound. To this end we first discover 200 parts that capture discriminative features in the training data. We then run graph cut on each training image to obtain part assignments to image regions. Each cell in the 10×10 image grid is a node in the graph. Two nodes in the graph are connected if their corresponding cells in the image grid are next to each other. Unary terms in the graph cut are the dot product scores between the feature vector extracted from an image region and a part filter plus the corresponding region-to-part assignment score. Pairwise terms in the graph cut implement a *Potts* model that encourages coherent labelings. Specifically, the penalty of labeling two neighboring nodes differently is λ and it is zero otherwise. λ controls the coherency of the initial assignments. We experiment using $\lambda \in \{0, 0.25, 0.5, 1\}$. We also experiment with random initialization, which corresponds to assigning z_i 's randomly. This is the simplest form of initialization and does not require discovering initial part filters.

We do G-MM optimization using both random and biased bounds. For the latter we use a bias function $g(b, w)$ that measures coherence of the labeling from which the bound was constructed. Recall from Equation 5.35 that each bound in $b \in B_t$ corresponds to a labeling of the image regions. We denote the labeling corresponding to the bound b by $z(b) = (z_1, \dots, z_n)$ where $z_i = (z_{i,1}, \dots, z_{i,100})$ specifies part assignments for all the 100 regions in the i -th image. Also, let $E(z_i)$ denote a function that measures coherence of the labeling z_i . In fact, $E(z_i)$ is the Potts energy function on a graph whose nodes are $z_{i,1}, \dots, z_{i,100}$. The graph respects a 4-connected neighborhood system (recall that $z_{i,r}$ corresponds to the r -th cell in the 10×10 grid defined on the i -th image). If two neighboring nodes $z_{i,r}$ and $z_{i,s}$ get different labels the energy $E(z_i)$

Opt. Method	Random		$\lambda = 0.00$		$\lambda = 0.25$		$\lambda = 0.50$		$\lambda = 1.00$	
	Acc.% \pm std	O.F.	Acc. %	O.F.	Acc. %	O.F.	Acc. %	O.F.	Acc. %	O.F.
CCCP	41.94 \pm 1.1	15.20	40.88	14.81	43.99	14.77	45.60	14.72	46.62	14.70
G-MM random	47.51 \pm 0.7	14.89	43.38	14.71	44.41	14.70	47.12	14.66	49.88	14.58
G-MM biased	49.34 \pm 0.9	14.55	44.83	14.63	48.07	14.51	53.68	14.33	56.03	14.32

Table 5.3: Performance of LS-SVM trained with CCCP and G-MM on MIT-Indoor dataset. We report classification accuracy (Acc.%) and the training objective value (O.F.). Columns correspond to different initialization schemes. “Random” assigns random parts to regions. λ controls the coherency of the initial part assignments: $\lambda = 1$ ($\lambda = 0$) corresponds to the most (the least) coherent case. “G-MM random” uses random bounds and “G-MM biased” uses the bias function of Equation 5.39. $\eta = 0.1$ in all the experiments. Coherent initializations lead to better models in general, but, they require discovering good initial parts. “G-MM” outperforms CCCP, especially with random initialization. “G-MM biased” performs the best. The value of the progress coefficient is set to $\eta = 0.1$ in these experiments.

increases by 1. For biased bounds we use the following bias function which favors bounds that correspond to more coherent labelings:

$$g(b, w) = - \sum_{i=1}^n E(z_i), \quad z(b) = (z_1, \dots, z_n). \quad (5.39)$$

Table 5.3 compares performance of models trained using CCCP and G-MM with random and biased bounds. For G-MM with random bounds we repeat the experiment five times and report the average over these five trials. Also, for random initialization, we do five trials using different random seeds and report the mean and standard deviation of the results. G-MM does better than CCCP under *all* initializations. It also converges to a solution with lower training objective value than CCCP. Our results show that picking bounds uniformly at random from the set of valid bounds is slightly (but consistently) better than committing to the CCCP bound. We get a remarkable boost in performance when we use a reasonable prior over bounds (*i.e.* the bias function of Equation 5.39). With $\lambda = 1$, CCCP attains accuracy of 46.6%, whereas G-MM attains 49.9%, and 56.0% accuracy with random and biased initialization respectively. Moreover, G-MM is less sensitive to initialization.

experiment	setup	MM	G-MM	
			random	biased
scene recognition	$\lambda = 0.0$	145	107	87
	$\lambda = 1.0$	65	69	138
data clustering (GMM-200)	forgy	35.76 ± 7.8	91.52 ± 4.4	
	rand. part.	114.98 ± 12.9	241.89 ± 2.1	
	k -means++	32.92 ± 5.8	80.78 ± 2.9	
data clustering (Cloud)	forgy	37.18 ± 12.1	87.68 ± 15.4	
	rand. part.	65.14 ± 18.7	138.64 ± 5.9	
	k -means++	21.3 ± 4.1	44.12 ± 10.7	

Table 5.4: Comparison of the number of iterations it takes for MM and G-MM to converge in the scene recognition and the data clustering experiment with different initializations and/or datasets. The numbers reported for the clustering experiment are the average and standard deviation over 50 trials.

5.5.4 Run Time

G-MM bounds make a fraction of the progress made in each bound construction step compared to the MM bound. Therefore, we would expect G-MM to take more steps before it converges. We report the number of iterations that MM and G-MM take to converge in Table 5.4. The results for G-MM depend on the value of the progress coefficient η which is set to match the experiments in the paper; $\eta = 0.02$ for the clustering experiment (Section 5.5.1) and $\eta = 0.10$ for the scene recognition experiment (Section ??).

The overhead of the bound construction step depends on the application. For example, in the scene recognition experiment, optimizing the bounds takes orders of magnitude more than sampling them (a couple of hours vs. a few seconds). In the clustering experiment, however, the optimization step is solved in closed form whereas sampling of the bounds involves performing a random walk on a large graph which can take a couple of minutes to run.

5.6 G-MM in Expectation (G-MME)

Although G-MM is more flexible and relaxed than MM, however, G-MM still requires greedily making progress in each step (see the second constraint in (5.3)). In this

section, we extend G-MM by allowing it to occasionally move to worse solutions, and thereby, standing a chance of getting out of local minima. The idea is to select a bound from \mathcal{F} at random from a distribution that guarantees making progress *in expectation* and not necessarily in *all* iterations. We call this method *G-MM in Expectation (G-MME)*.

Let \mathcal{P} denote the set of all probability distributions defined on \mathcal{F} ; that is

$$\mathcal{P} = \left\{ P : \mathcal{F} \rightarrow \mathbb{R} \mid P(f) \geq 0, \forall f \in \mathcal{F}, \text{ and } \sum_{f \in \mathcal{F}} P(f) = 1 \right\}. \quad (5.40)$$

G-MME selects bounds by sampling from a distribution $P_t \in \mathcal{P}$. Let b denote a random variable with probability distribution P_t . In order to *make progress in expectation* G-MME requires that

$$\mathbb{E}_b[b(w_{t-1})] \leq b_{t-1}(w_{t-1}) - \eta d_t \quad (5.41)$$

where $d_t = b_t(w_t) - F(w_t)$ is the gap between the bound selected in iteration t and the objective function at w_t , and $\eta \in (0, 1]$ is the progress coefficient.

G-MM guarantees making progress by adjusting the set of valid bounds over time (see (5.6)). In G-MME, however, $\mathcal{B}_t = \mathcal{F}$ at all times, and making progress is guaranteed by adjusting the distribution P_t over time. Since, *all* bounds in \mathcal{F} stand a chance of getting selected in each iteration, G-MME always have a chance of getting out of local minimum. Whereas, in G-MM, if at some iteration the minimizers of all valid bounds are within the attraction basin of a local minimum, the method is destined to eventually converge to that local minimum.

5.6.1 Sampling Bounds from Maximum Entropy Distribution

The principle of maximum entropy states that, among all possible probability distributions, the distribution with the highest entropy makes the least assumptions and is the closest distribution to uniform. In G-MME we define P_t to be the maximum

entropy distribution among the distributions in \mathcal{P} that satisfy (5.41):

$$\begin{aligned}
P_t &= \arg \max_{P \in \mathcal{P}} H(P) \\
&s.t. \mathbb{E}_{b \sim P}[b(w_{t-1})] \leq b_{t-1}(w_{t-1}) - \eta d_t.
\end{aligned} \tag{5.42}$$

It can be shown that the maximum entropy distribution P_t is an exponential distribution of the following form for some scalar λ :

$$P_\lambda(b) = \frac{1}{Z_\lambda} e^{\lambda b(w_{t-1})}, \quad Z_\lambda = \sum_{f \in \mathcal{F}} e^{\lambda f(w_{t-1})}, \tag{5.43}$$

and we can rewrite the optimization problem of (5.42) as follows:

$$\begin{aligned}
P_t &= P_{\lambda^*} \text{ where } \lambda^* = \arg \max_{\lambda} H(P_\lambda) \\
&s.t. \mathbb{E}_{b \sim P_\lambda}[b(w_{t-1})] \leq b_{t-1}(w_{t-1}) - \eta d_t.
\end{aligned} \tag{5.44}$$

λ controls the skewness of the distribution. In particular, $\lambda = 0$ corresponds to the uniform distribution and $\lambda = -\infty$ corresponds to the distribution that has all its mass on $\arg \min_{b \in \mathcal{F}} b(w_{t-1})$.

We prove two theorems (5.4 and 5.5) that make it possible to do binary search on the value of λ in order to find the distribution P_t that solves (5.44). In the rest of this section we define $h(b) \equiv b(w_{t-1})$ and $\mathbb{E}[\cdot] \equiv \mathbb{E}_{b \sim P}[\cdot]$ to simplify notation.

Theorem 5.4. *If $P_\lambda(b) \propto e^{\lambda h(b)}$, then $H(P_\lambda)$ is a decreasing function of $|\lambda|$.*

Proof. We show that $\frac{\partial H}{\partial \lambda}$ is positive when $\lambda < 0$ and negative when $\lambda > 0$.

$$\begin{aligned}
H(P) &= - \sum_{b \in \mathcal{F}} P(b) \log(P(b)) \\
&= - \sum_{b \in \mathcal{F}} \frac{e^{\lambda h(b)}}{Z_\lambda} (\lambda h(b) - \log Z_\lambda) \\
&= \frac{-\lambda}{Z_\lambda} \sum_{b \in \mathcal{F}} e^{\lambda h(b)} h(b) + \log Z_\lambda \\
\Rightarrow \frac{\partial H}{\partial \lambda}(P) &= \frac{-Z_\lambda + \lambda \sum_{b \in \mathcal{F}} h(b) e^{\lambda h(b)}}{Z_\lambda^2} \sum_{b \in \mathcal{F}} e^{\lambda h(b)} - \lambda \mathbb{E}[h^2(b)] + \frac{\sum_{b \in \mathcal{F}} h(b) e^{\lambda h(b)}}{Z_\lambda} \\
&= -\mathbb{E}[h(b)] + \lambda \mathbb{E}[h(b)]^2 - \mathbb{E}[h^2(b)] + \mathbb{E}[h(b)] \\
&= -\lambda \operatorname{var}(h(b)) \begin{cases} > 0 & \text{if } \lambda < 0 \\ < 0 & \text{if } \lambda > 0 \\ = 0 & \text{if } \lambda = 0 \end{cases} \tag{5.45}
\end{aligned}$$

□

Theorem 5.5. *If $P_\lambda(b) \propto e^{\lambda h(b)}$, then $\mathbb{E}_{b \sim P_\lambda}[h(b)]$ is an increasing function of λ unless $h(b)$ is the same for all $b \in \mathcal{F}$ where $h(b) = b(w_{t-1})$.*

Proof. We show that $\frac{\partial \mathbb{E}_{b \sim P}[h(b)]}{\partial \lambda}$ is non-negative for any λ .

$$\begin{aligned}
\mathbb{E}_{b \sim P}[h(b)] &= \sum_{b \in \mathcal{F}} P(b)h(b) \\
\frac{\partial \mathbb{E}[h(b)]}{\partial \lambda} &= \sum_{b \in \mathcal{F}} \frac{\partial P}{\partial \lambda}(b)h(b) \\
\frac{\partial P}{\partial \lambda}(b) &= \frac{\frac{\partial e^{\lambda h(b)}}{\partial \lambda} Z_\lambda - \frac{\partial Z_\lambda}{\partial \lambda} e^{\lambda h(b)}}{Z_\lambda^2} \\
&= \frac{(h(b)e^{\lambda h(b)})(Z_\lambda) - \left(\sum_{f \in \mathcal{F}} h(f)e^{\lambda h(f)}\right)(e^{\lambda h(b)})}{Z_\lambda^2} \\
&= P(b)\left(h(b) - \mathbb{E}[h(b)]\right) \tag{5.46}
\end{aligned}$$

$$\begin{aligned}
\Rightarrow \frac{\partial \mathbb{E}[h(b)]}{\partial \lambda} &= \sum_{b \in \mathcal{F}} P(b)\left(h(b) - \mathbb{E}[h(b)]\right)h(b) \\
&= \sum_{b \in \mathcal{F}} P(b)h^2(b) - \mathbb{E}[h(b)] \sum_{b \in \mathcal{F}} P(b)h(b) \\
&= \mathbb{E}[h^2(b)] - \mathbb{E}^2[h(b)] = \text{var}(h(b)) \geq 0 \tag{5.47}
\end{aligned}$$

□

Theorems 5.4 and 5.5 imply that we can do binary search on $\lambda \in (-\infty, 0]$ to solve (5.44). This assumes the expectation $\mathbb{E}_{b \sim P_\lambda}[b(w_{t-1})]$ can be computed efficiently. More precisely, we start by setting $\lambda_{low} = -A$ and $\lambda_{high} = 0$, where A is a large positive value. We set $\lambda = \frac{\lambda_{low} + \lambda_{high}}{2}$ and check if $\mathbb{E}_{b \sim P_\lambda}$ satisfies the constraint in (5.44); if so, we set $\lambda_{low} = \lambda$, otherwise, we set $\lambda_{high} = \lambda$. We keep doing this until $\lambda_{high} - \lambda_{low} < \epsilon$, where ϵ is a small positive value. When this happens we have $\lambda^* \approx \lambda$.

After finding λ^* , the G-MME bound in iteration t is obtained by sampling from $P_t = P_{\lambda^*}$ i.e. $b_t \sim P_t$. The hardness of the sampling procedure depends on the form of \mathcal{F} and b . For example, in the clustering example of Section 5.4.1 bounds correspond to fixing the assignment of the data points to cluster centers (see (5.31)) and for a bound b corresponding to the assignment vector $z = (z_1, \dots, z_n)$ we have

$$h(b) = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2, \quad w_{t-1} = (\mu_1; \dots; \mu_k). \tag{5.48}$$

In this particular case z_i s are independent and we can draw samples from P_t simply by sampling z_i s from the discrete distribution $P_i(j) \propto e^{\lambda^* \|x_i - \mu_j\|^2}$ for $1 \leq j \leq k$.

5.7 Conclusion

We introduced Generalized Majorization-Minimization (G-MM), a generic iterative bound optimization framework that generalizes upon Majorization-Minimization (MM). Our key observation is that MM enforces an overly-restrictive touching constraint in its bound construction step, making it inflexible and sensitive to initialization. G-MM relaxes this constraint by adopting a different measure of progress than MM. As a result, there are multiple valid bounds that can be selected in each iteration of the optimization process. We propose deterministic and stochastic ways of selecting bounds in each step. This generalized bound construction process leads to optimization methods that are less sensitive to initialization, and enjoy the ability to directly incorporate rich application-specific priors and constraints, without the need for modifying the true objective function. Experimental result with several latent variable models show that G-MM algorithms significantly outperform their MM counterparts.

We extend G-MM further and allow it to occasionally select bounds that do not make progress, and thereby, stand a chance of getting out of imminent local optima. More precisely, we require the bound in each step to make progress *in expectation*, and refer to the resulting model as *G-MM in Expectation (G-MME)*.

Future work includes applying G-MM(E) to a wider range of problems and theoretical analysis, such as convergence rate. For example, an idea similar to G-MME can be applied to gradient descent method. More precisely, in each iteration, we can move in a *random* direction drawn from a maximum entropy distribution such that, *in expectation*, the value of the objective function is guaranteed to improve.

Chapter 6

Conclusion

This thesis focuses on the problem of image classification in presence of latent variables. We proposed latent variable models (LVMs) to solve different tasks in computer vision including scene recognition and object detection. Training latent variable models often entail optimizing a non-convex objective function. We proposed new optimization techniques that are useful for training latent variable models and other non-convex optimization problems.

Reconfigurable Bag of Words (RBoW) model [47] is a latent variable model for scene recognition that represents a scene as a collection of parts arranged in a reconfigurable pattern. RBoW partitions an image into a pre-defined set of regions and uses latent variables to assign one part to each image region. Each part can be thought of as a region type, such as *sky*, *building*, *grass*, etc., and inference provides a semantic segmentation of the image in addition to the scene category prediction. Image regions in RBoW are very coarse (16 cells of a 4×4 spatial grid on the image), and using smaller image regions degrades the performance because RBoW assumes image regions are independent conditional on the image category label. We extended RBoW by introducing pairwise dependencies between image regions. For example, the new model can encourage coherent semantic labeling of the image regions, and can penalize certain geometric configurations, such as *water on top of sky*. Inference with the proposed model is NP-hard, and we used the α/β -swap algorithm [9] together with graph-cut to find an approximate solution to the inference problem.

Another way to do image classification using part-based models is to represent an image as a vector of part responses, and train a classifier for each category on this shared representation. The location of each part in the image is treated as a latent variable in this model. We presented an algorithm for training all the parameters in such models jointly, including the set of shared part filters and the set of category specific classifiers. Unlike other existing approaches, that use hand-engineered heuristics to discover distinctive parts, all stages in our training pipeline are driven by the same objective namely the classification performance on the training set. We showed that this part-based model can be seen as a 3-layer neural network with a convolutional layer that computes the part responses at each image location, followed by a max-pooling layer, and a fully connected layer that computes the category specific classification scores. While neural networks are typically trained using (stochastic) gradient descent (back-propagation, in particular), we train our model using sequential convex optimization which is deterministic, more reliable, and easier to work with in practice. For example, the quality of the solution in our training algorithm does not depend on careful tuning of hyper-parameters such as learning rate schedule.

We also introduced an extension to the family of LVMs, called Generalized LVM (GLVM). The latent variables in an LVM, referred to as *positive latent variables*, collect evidence for the presence of the foreground object in the image. In addition to positive latent variables, GLVM has a special type of latent variables, referred to as *negative latent variables*, that collect counter evidence for presence of the foreground object. Negative latent variables can make it easier for the classifier to distinguish between the foreground object and other similar objects in the image or the background by firing on parts of the other objects. For example, a *cow* detector can use a negative latent variable that captures a *saddle* to distinguish a *cow* from a *horse*.

All the latent variable models that we studied in this thesis lead to a non-convex training objective. Existing non-convex optimization methods that work by alternating between constructing a bound on the objective function and optimizing the bound, such as Expectation Maximization (EM) and Concave-Convex Procedure (CCCP), are observed to be sensitive to initialization. We argue that this is due to an unnec-

essary and overly-restrictive constraint that requires the bound at each iteration to touch the objective function at the solution of the previous iteration. We proposed a generic framework for optimizing non-convex objectives, called G-MM, which relaxes this constraint. Moreover, we proved that, under mild conditions, G-MM converges to a local optimum or a saddle point of the true objective. Our experimental result shows that G-MM tends to converge to a better solution compared to EM and CCCP, and is also less sensitive to initialization.

Bibliography

- [1] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Symposium on Discrete Algorithms (SODA)*, 2007.
- [2] Mathieu Aubry, Bryan Russell, and Josef Sivic. Painting-to-3D model alignment via discriminative visual elements. *ACM Transactions on Graphics*, 2013.
- [3] Hossein Azizpour, Mostafa Arefiyan, Sobhan Naderi Parizi, and Stefan Carlsson. Spotlight the negatives: A generalized discriminative latent model. In *BMVC*, 2015.
- [4] Hakan Bilen, Marco Pedersoli, and Tinne Tuytelaars. Weakly supervised object detection with posterior regularization. In *British Machine Vision Conference (BMVC)*, 2014.
- [5] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [6] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Scene classification using a hybrid generative/discriminative approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2008.
- [7] L eon Bottou. Stochastic gradient tricks. *Lecture Notes in Computer Science*, 2012.
- [8] Lubomir D. Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2009.
- [9] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2001.
- [10] Ayan Chakrabarti, Ying Xiong, Steven Gortler, and Todd Zickler. Low-level vision by consensus in a spatial hierarchy of regions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [11] Ramazan G. Cinbis, Jakob Verbeek, and Cordelia Schmid. Multi-fold MIL training for weakly supervised object localization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

- [12] Timothy Cootes, Gareth Edwards, and Christopher Taylor. Comparing active shape models with active appearance models. In *BMVC*, 1999.
- [13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Journal of Machine Learning*, 2009.
- [14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [15] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1977.
- [16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [17] Carl Doersch, Abhinav Gupta, and Alexei Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, 2013.
- [18] Ian Endres, Kevin Shih, Johnston Jiaa, and Derek Hoiem. Learning collections of part models for object recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [19] Li Fei-Fei and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [20] Pedro Felzenszwalb, Ross Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2010.
- [21] Pedro Felzenszwalb and David McAllester. Object detection grammars. In *Technical report*, 2010.
- [22] Ross Girshick Pedro Felzenszwalb and David McAllester. Object detection with grammar models. In *NIPS*, 2011.
- [23] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transaction on Knowledge Discovery from Data (KDD)*, 2007.
- [24] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [25] Ross Girshick and Jitendra Malik. Training deformable part models with decorrelated features. In *ICCV*, 2013.

- [26] Bharath Hariharan, Jitendra Malik, and Deva Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, 2012.
- [27] Jeremy Heitz, Gal Elidan, Benjamin Packer, and Daphne Koller. Shape-based object localization for descriptive classification. *International journal of computer vision (IJCV)*, 2009.
- [28] David Hunter, Kenneth Lange, and Ilsoon Yang. Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, 2000.
- [29] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.
- [30] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 2009.
- [31] Mayank Juneja, Andrea Vedaldi, C. V. Jawahar, and Andrew Zisserman. Blocks that shout: Distinctive parts for scene classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [32] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2004.
- [33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.
- [34] Pawan Kumar, Ben Packer, and Daphne Koller. Modeling latent variable uncertainty for loss-based learning. In *International Conference on Machine Learning (ICML)*, 2012.
- [35] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.
- [36] Bastian Leibe, Ales Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *IJCV*, 2008.
- [37] Li-Jia Li, Hao Su, Eric Xing, and Li Fei-Fei. Object bank: A high level image representation for scene classification and semantic feature sparsification. In *NIPS*, 2010.
- [38] Ivan Lillo, Alvaro Soto, and Juan Carlos Niebles. Discriminative hierarchical modeling of spatio-temporally composable human activities. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

- [39] Pamela Lipson, Eric Grimson, and Pawan Sinha. Configuration based scene classification and image indexing. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997.
- [40] Hans Lobel, Rene Vidal, and Alvaro Soto. Hierarchical joint max-margin learning of mid and top level representations for visual recognition. In *ICCV*, 2013.
- [41] David Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.
- [42] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, 1967.
- [43] Radford Neal and Geoffrey Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, 1998.
- [44] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001.
- [45] Megha Pandey and Svetlana Lazebnik. Scene recognition and weakly supervised object localization with deformable part-based models. In *ICCV*, 2011.
- [46] Devi Parikh and Kristen Grauman. Relative attributes. In *ICCV*, 2011.
- [47] Sobhan Naderi Parizi, John Oberlin, and Pedro Felzenszwalb. Reconfigurable models for scene recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [48] Sobhan Naderi Parizi, Andrea Vedaldi, Andrew Zisserman, and Pedro Felzenszwalb. Automatic discovery and optimization of parts for image classification. In *ICLR*, 2015.
- [49] Omkar Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [50] Wei Ping, Qiang Liu, and Alexander Ihler. Marginal structured svm with hidden variables. In *International Conference on Machine Learning (ICML)*, 2014.
- [51] Hamed Pirsiavash and Deva Ramanan. Parsing videos of actions with segmental grammars. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [52] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [53] Mohammad Rastegari, Hannaneh Hajishirzi, and Ali Farhadi. Discriminative and consistent similarities in instance-level multiple instance learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

- [54] Nima Razavi, Juergen Gall, Pushmeet Kohli, and Luc Van Gool. Latent hough transform for object detection. In *ECCV*, 2012.
- [55] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR Deep Vision workshop*, 2014.
- [56] Christian Ries, Fabian Richter, and Rainer Lienhart. Towards automatic bounding box annotations from weakly labeled images. *Multimedia Tools and Applications*, 2015.
- [57] Mohammad Amin Sadeghi and Ali Farhadi. Recognition using visual phrases.
- [58] Mark Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab.
- [59] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *International Conference on Machine Learning (ICML)*, 2007.
- [60] Saurabh Singh, Abhinav Gupta, and Alexei Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012.
- [61] Hyun Oh Song, Ross Girshick, Stefanie Jegelka, Julien Mairal, Ziad Harchaoi, and Trevor Darrell. On learning to localize objects with minimal supervision. In *International Conference on Machine Learning (ICML)*, 2014.
- [62] Xi Song, Tianfu Wu, Yunde Jia, and Song-Chun Zhu. Discriminatively trained and-or tree models for object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [63] Jian Sun and Jean Ponce. Learning discriminative part detectors for image classification and cosegmentation. In *ICCV*, 2013.
- [64] Meng Tang, Ismail Ben Ayed, and Yuri Boykov. Pseudo-bound optimization for binary energies. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.
- [65] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms.
- [66] Cor Veenman, Marcel Reinders, and Eric Backer. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2002.
- [67] Shuo Wang, Yizhou Wang, and Song-Chun Zhu. Hierarchical space tiling for scene modeling. In *ACCV*, 2012.

- [68] Jianxin Wu and James Rehg. Centrist: A visual descriptor for scene categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2010.
- [69] Oksana Yakhnenko, Jakob Verbeek, and Cordelia Schmid. Region-based image classification with a latent svm model. *INRIA Tech. Rep. 7665*, 2011.
- [70] Weilong Yang, Yang Wang, Arash Vahdat, and Greg Mori. Kernel latent svm for visual recognition. In *NIPS*, 2012.
- [71] Chun-Nam Yu and Thorsten Joachims. Learning structural svms with latent variables. In *ICML*, 2009.
- [72] Alan Yuille and Anand Rangarajan. The concave-convex procedure. 2001.
- [73] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.
- [74] Jun Zhu, Li-Jia Li, Fei-Fei Li, and Eric Xing. Large margin training of upstream scene understanding models. In *NIPS*, 2010.
- [75] Jun Zhu, Tianfu Wu, Song-Chun Zhu, Xiaokang Yang, and Wenjun Zhang. Learning reconfigurable scene representation by tangram model. In *WACV*, 2012.
- [76] Jun-Yan Zhu, Yong Jae Lee, and Alexei A Efros. Averageexplorer: Interactive exploration and alignment of visual data collections. In *SIGGRAPH*, 2014.
- [77] Long Zhu, Yuanhao Chen, and Alan Yuille. Learning a hierarchical deformable template for rapid deformable object parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2010.