Optimization for and by Machine Learning

Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Engineering

by Pritish Mohapatra

201299645

`pritish.mohapatra@research.iiit.ac.in`

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500032, INDIA
December 2021

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Optimization for and by Machine Learning" by Pritish Mohapatra, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. C.V. Jawahar
Date	Adviser: Prof. M. Pawan Kumar

To

My parents

Acknowledgement

First and foremost, I would like to take this opportunity to thank Prof. C.V. Jawahar and Dr. M. Pawan Kumar for their guidance as my Phd advisors. Their passion for research was a valuable source of inspiration for me as a student. This work immensely benefited from their deep understanding of the area of machine learning, optimization and computer vision. Apart from insightful technical discussions, their advice with regards to research methodology and technical writing were invaluable for my Phd.

I would like to thank my collaborators Aseem, Puneet and Michal whom I had the opportunity to work with on different parts of this work. It was a wonderful experience to conduct research with such talented fellow student researchers.

I deeply appreciate the effort put in by the faculty members and the administrative staff at the Center for Visual Information Technology (CVIT, IIIT-Hyderabad) in maintaining a healthy academic environment that facilitated research and innovation. At various points in my PhD, I received valuable support and encouragement from Prof. P.J. Narayanan, Prof. J. Sivaswamy and Dr. A. Namboodiri at CVIT.

I am grateful to Tata Consultancy Services for providing me with generous financial support in the form of the Phd research scholarship program.

With the warmest thanks, I appreciate all my lab mates for their constant support during my PhD. I thank Arunava, Aniket, Vijay, Nataraj, Anand, Praveen and Aditya for all the elongated yet interesting discussions on the blackboard; Saurabh, Rajvi and Parikshit for making me feel welcomed in their conversations around computer vision, graphics and research in general; and Devendra and Govinda for all the spontaneous enthusiastic discussions into topics in machine learning and mathematics.

Lastly, I thank my family for believing in me and for the patient support throughout the years. Their optimism and encouragement was of immense help to push through the tough times.

Abstract

In machine learning, tasks like making predictions using a model and learning model parameters can often be formulated as optimization problems. The feasibility of using a machine learning model depends on the efficiency with which the corresponding optimization problems can be solved. As such, the area of machine learning throws up many challenges and interesting problems for research in the field of optimization. While in some cases, it is possible to directly apply off-the-shelf optimization methods for problems in machine learning, in many other cases, it becomes necessary to develop optimization algorithms that are tailor-made for specific problems. On the other hand, developing optimization algorithms for specific problem domains can itself be helped by machine learning techniques. Learning optimization algorithms from data can help relieve tedious effort required to develop optimization methods for new problem domains. The challenge here is to appropriately parameterize the space of algorithms for different optimization problems. In this context, we explore the interplay between the areas of optimization and machine learning and make contributions in specific problems of interest that lie in the overlap of these fields.

We look into the optimization challenge presented by multi-class classification with a large number of output classes and propose a partial-linearization based approach that intuitively generalizes over several popular optimization algorithms applicable to this task. It is popular to use measures like average precision (AP) and normalized discounted cumulative gain (NDCG) to evaluate a model for a classification task or a ranked retrieval task, however, they are not as popularly used for learning the parameters of these models. This is because of the difficulty in optimizing these non-decomposable loss functions. We propose a useful characterization of a large class of such loss functions that we show are amenable to efficient optimization and present an algorithm that can be used to efficiently optimize this class of loss functions. On the other hand, we explore the possible application of machine learning techniques for developing optimization algorithms, specifically, for combinatorial optimization problems. A wide class of approximate algorithms for NP-hard combinatorial optimization problems solve a continuous relaxation of the original problem and then round the fractional solution to obtain an approximate discrete solution. The rounding procedures involved in such methods are non-trivial and generally have to be ingeniously designed for each task. We propose a framework that allows the learning of such rounding procedures from unsupervised data. Such a learning based approach permits rapid development of algorithms for novel optimization problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization as a method for problem solving is ubiquitous across all areas of human interest. Intuitively, optimization involves selecting an element from a set of options that is the best or worst in a certain predefined way. Mathematically, a wide range of optimization problems can be formulated as the task of finding an element that minimizes or maximizes a real valued function over an allowed set. A large number of optimization problems of interest differ mainly in the type of the function that has to be minimized or maximized and nature of the set of allowed elements. Several different forms of this problem routinely come up in many real world scenarios like in the areas of operations research and engineering.

Over the course of its development, optimization methods have progressed through phases that employed a diverse set of approaches. Early developments in optimization methods can be traced back to ideas from differential calculus developed by Newton and Leibniz and those from calculus of variations developed by Euler and Bernoulli. Over time the application of the Cauchy's steepest descent method for unconstrained optimization and Lagrange methods for constrained optimization gained wide popularity. More recently with the advent of computers and increase in compute capabilities led to rapid development in optimization algorithms. Focus towards developing efficient methods catering to specific forms of optimization problems, which saw impact-making developments like Dantzig's Simplex algorithm [20] for Linear programs and Karmarkar's interior point algorithm [50] for convex programs, have since led to increased effort towards further specialization of optimization algorithms into new classes like semi-definite programming (SDP) [96] and second order cone programming (SOCP) [61]. While a great deal of research in optimization methods can be associated with branching and specialization of the field based on theoretical structure of the problems, a substantial body of research has also been motivated and directed by the several application domains. For example, operations research, circuit design, signal processing and artificial intelligence [11]. Of special interest to us here is the interaction between the fields of optimization methods and machine learning. Machine learning has not only been a successful area of application for optimization algorithms but has also motivated new directions of research in optimization methods [7, 87].

Machine learning is concerned with the task of making a machine learn from experience. Specifically, learning involves searching for a model of an environment or a task based on observations, with the goal of making predictions for novel queries for that task. In a more formal sense, learning can be imagined to be a sophisticated form of curve-fitting that tries to fit an appropriate model or function to a collection of data points. Such a curve-fitting problem can be generally posed as an optimization problem. For example, when trying to make a machine learn to play the board game of Go, the goal is to build a model of the game that increases the chances of winning and this is done by optimizing for the win based reward.

In the early days of machine learning, limited access to data and low compute capabilities encouraged focus on models that catered to specific input data structures. This allowed the models to be designed specifically to leverage the structure in the specific type of data and to be tuned for any particular task with a small amount of data. Such focus on a specific problem or data structure allowed for thorough theoretical analysis of the models and as a result for theoretical guarantees on the performance of the models like support vector machines (SVMs) [17], decision forests [51] and boosting [82]. While in some cases off-the-shelf optimization algorithms can be employed to search through the hypothesis space for the optimal model, it is sometimes necessary to design novel optimization methods tailor-made for the specific problem structure. This has been the motivation for a wide range of optimization algorithms that have been developed to facilitate efficient learning in different domains.

On the other hand, with the availability of huge amount of data and compute capability, it has gradually become possible to use more complex and flexible machine learning models like deep neural nets. While these models required more data and computational resources to properly fit to the task and data at hand, on the upside, similar models could now be used for multiple different domains without having to modify them too much for any specific domain. Such models while being difficult to theoretically analyse (and therefore difficult to provide theoretical guarantees for performance), have been shown to perform much better empirically for many real world settings. Because of the highly complex nature of the hypothesis space represented by models like the deep neural networks, theoretical analysis of the corresponding optimization algorithms is very difficult. This has led to increased attention being given to the practical efficiency and scalability of optimization methods being developed for training such models rather than focusing too much on the theoretical guarantees.

While optimization methods form an integral part of machine learning pipelines, there is increasing realization that learning techniques can also be used to estimate certain intermediate quantities or mappings that can be of help in solving an optimization problem. In this context, it is of interest to explore the mutual relevance and usefulness of optimization methods and learning systems. We elaborate on the interplay between optimization and learning in the following sections.

1.1 Optimization Problems in Machine Learning

In past few decades, there has been a fundamental shift in the approach humans have taken to designing intelligent systems. Instead of trying to mimic the decision making process of humans, the machines are now designed to mimic the learning process of humans. Such a machine learning framework comprises a model with parameters that are to be learned from appropriate task specific data presented to it. The explosion in data that we have been witnessing since the last decade, along with the rapid developments in computational hardware has massively increased the viability and effectiveness of such methods. Such an approach has also allowed development of intelligent systems for incredibly complex tasks with minimal design effort from a human expert.

In a formal setting, let say we have a task at hand that requires us to predict the value of variable y which depends on the input x. For instance, x can be an MRI scan of a tumor and y can correspond to a binary variable that states if the tumor is benign or malignant. Applying machine learning to such a task generally involves two different computational problems. If we can assume that $y = f(x;\theta)$, where f is the function that maps the input to the output and has parameters θ, then the first computational problem is to compute output y by evaluating function $f(x;\theta)$ at input x. This is called the *prediction problem* or the *inference problem*. Second, given some data $\mathcal{D}$, the parameters θ of the function f can be estimated as $\theta^* = \operatorname{argmin}_\theta L(f, \mathcal{D})$, where L is an appropriate task specific loss function that measures the quality of a parameter configuration. This is called the *learning problem*.

Generally, the inference problem and the learning problem in machine learning can be seen as optimization problems. The inference problem often involves selecting the candidate output y from a allowed set $\mathcal{Y}$ that maximizes a certain *score* that estimates the quality of a prediction:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}} \; score(x, y). \tag{1.1}$$

The domain set $\mathcal{Y}$ varies for every task. While it would be a categorical set for a classification problem, it would be a continuous space for a regression task. The domain set can sometimes be a more sophisticated structured set like that of all possible rankings of a set of elements as in the case of ranked retrieval tasks. While it is possible to deploy off-the-shelf optimization algorithms for solving many such inference problems, certain cases demand developing novel optimization methods that can leverage the structure of the problem to solve the inference problem effectively. In many cases, the design of the scoring function is influenced by its pliability for efficient optimization.

In a standard setting of the learning problem, the parameters θ of a machine learning model $f(x;\theta)$ are chosen such that the model minimizes the expected value of the loss function $L(f(x;\theta),y)$ over the data distribution P_{data}:

$$\theta^* = \operatorname*{argmin}_\theta E_{(x,y)\sim P_{data}}\left[L(f(x;\theta),y)\right]. \tag{1.2}$$

The objective function of the above optimization problem is called *risk*. Computing risk requires the knowledge of the data distribution P_{data} which we generally don't know exactly. Therefore, we instead optimize an empirical estimate of the expectation of the loss function on the training data $\mathcal{D}_{train}$ which is called the *empirical risk*:

$$\theta^* = \operatorname*{argmin}_\theta \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(f(x;\theta),y). \tag{1.3}$$

While the model is trained only on the training set, it is hoped that it also performs well on any unseen data from the data distribution. The generalization performance of a model can be evaluated by computing the risk on a test set $\mathcal{D}_{test}$ that is disjoint from the training set. In most cases, the learning algorithm tends to over-fit the model onto the training set, thus harming the generalization performance. This is generally dealt with by adding an additional *regularization* term to the objective function in equation 1.3.

Many popular techniques in the optimization literature are directly applicable to solving the learning problem in machine learning. At the same time however, direct application of generic optimization solvers might be highly inefficient for certain cases. This can be because of the large scale of the problem, in terms of the size of the dataset or output space involved, or can be because of the complexity of the problem, in terms of the output space or the loss function. In order to deal with the large size of the dataset, stochastic variants of optimization methods that work only with a subset of the dataset at a time, have become popular. On the other hand, as in the case of inference, dealing with the complexity in the output space or the loss functions involved in the learning problem requires developing methods that can leverage the structure specific to the task to make the optimization efficient. An interested reader might want to refer to [76, 87] for a general overview of optimization problems and methods that are relevant to machine learning. In the following sections, we will look at some specific optimization challenges that are commonly encountered in machine learning systems and also discuss some approaches to deal with them.

1.2 Learning for Optimization

Optimization algorithms can generally be seen as iterative search methods trying to find an optimal element in the allowed set. In case of a continuous domain, each step of the algorithm generally involves choosing an appropriate update direction and then taking a step in that direction. The update direction and sometimes even the step size are generally a function of the objective function value or derivative at the current or previously encountered points in the search space. The various optimization algorithms over continuous domain differ in their choice of the update direction and the update step. Quite often it makes sense to mould this update computation to fit the specific problem structure. Here in lies the scope for using machine learning techniques to design appropriate update steps. In some cases, evaluating the function value or derivative of the objective function can itself be non-trivial and may require complex estimation techniques like variational approximation or sampling. Here again, learning based methods can be used to help come up with good approximation functions and samplers.

In a more general sense, most optimization algorithms can be imagined to be algorithms that involve sequential decision making. Apart from the continuous-domain optimization methods discussed above, combinatorial optimization algorithms that deal with discrete domains, also generally proceed by taking a sequence of decisions. For instance, a greedy algorithm to solve the graph based travelling salesman problem would involve making a series of locally optimal decisions. Many such combinatorial optimization problems happen to be NP-hard and the most popular solvers are often heuristic based. While traditionally such clever heuristic algorithms have been designed by human experts, there is scope to model these sequential algorithms as machine learning problems and try to learn them using data. In the subsequent sections, we will look into some interesting problems in this domain and discuss different approaches towards solving them.

1.3 Problems of interest

In context of the increasingly intertwined domains of optimization and machine learning, there are several problems that pose challenging questions to researchers. In this section, we shall look into some of the interesting problems that lie in the junction of these two computational fields.

1.3.1 Large Scale Classification Problems

Many tasks in the real world can be formulated as multi-class classification problems. In other words, given any object like an image, a video, a document or a speech sample, the task is to assign it a label that belongs to a specified finite set. For example, in the case of object recognition in an image, the label can be car, chair or person. Similarly, for action recognition from a video, actions categories like jumping, kicking or clapping can be candidate labels. Given an input $\mathbf{x} \in \mathcal{X}$ the aim of multi-class classification is to predict the output y that belongs to the output space $\mathcal{Y}$. If the number of classes is denoted by c, the output space $\mathcal{Y} = \{1, \ldots, c\}$.

The binary classification problem which deals with only two classes is a much simpler problem to solve, with a plethora of classical solutions like the Support Vector Machine (SVM), Perceptron and Logistic Regression [10]. There have been several works which have presented extensions of these methods to a multi-class setting [19, 27, 64, 79]. A popular thread of approaches to deal with a large number of output classes has been to decompose the multi-class classification task into classification problems that have to deal with smaller subsets of the bigger output space, most often binary. The multi-class classification problem then can be performed by solving a series of binary classification problems instead. Binary classifiers can be used to perform one-vs-one or one-vs-all classifications and the results can then be aggregated to make a multi-class classification [41]. Another set of methods try to partition the output space into individual classes by using a large number of binary decision boundaries. The output space can be partitioned using a decision tree or a directed acyclic graph [78] and such a partitioning can be informed by prior knowledge about structural relationship among the output classes.

While the above idea of decomposing the multi-class classification problem into a series of binary classification problems presents a simple yet powerful approach, such methods can not generally model the correlations between output classes unless they are explicitly encoded into the model. On the other hand, there are methods like multi-class SVMs [19] and neural networks which try to learn a single complex decision boundary that distinguishes among all the output classes. In doing so, these models are able to capture the correlation that the different output classes may have among them. However, learning the parameters for those models generally involve solving a single complex optimization problem that includes all the output classes. The complexity of the optimization problem rapidly rises with increase in the number of output classes and can become practically infeasible for several real world problems.

In order to make the optimization problem for learning a multi-class classification model feasible for a large output space, it is common to opt for simple learning objectives like the cross-entropy loss applied on softmax output scores. While computing the output scores require normalization over all the output classes and it can become very inefficient when the number of output classes are very large, it is still the most popular objective function that is used to train models like neural networks for multi-class classification. On the other hand, using more sophisticated learning criteria like margin-maximization can potentially give better classification performance [89]. The optimization problem that comes up in margin-maximization frameworks are however more complicated and can many times deter their usage for training large scale multi-class classification models. Investigation into efficient optimization methods for learning of large classification models in context of the margin-maximization framework is an important and interesting problem.

1.3.2 Optimization of Rank-based Loss Functions

The loss function involved in the training and evaluation of a machine learning model should ideally correspond to the performance of the model for a particular task. For example, in most cases of regression, we can use mean squared error as a loss function and optimize the corresponding empirical risk to learn the model parameters. Similarly in case of classification, 0-1 loss, that assigns a loss of 1 to every misclassification and 0 to every correct classification, is a sensible loss function. In case of tasks like classification and regression, the popular loss functions are generally an aggregate of loss computed over each individual sample. For example, the mean squared error is an average of the squared errors for individual samples. Similarly, the 0-1 loss is the average over individual misclassifications. Even though 0-1 loss might not be differentiable, there are good differentiable surrogate loss functions that are additively decomposable onto individual samples. The additive decomposablilty property allows for efficient optimization as the optimization problem can then be broken down into sub-problems over individual samples.

In case of other tasks like that of information retrieval which often involves ranking the samples in a dataset, most popular evaluation measures are a non-decomposable function of the entire dataset. For

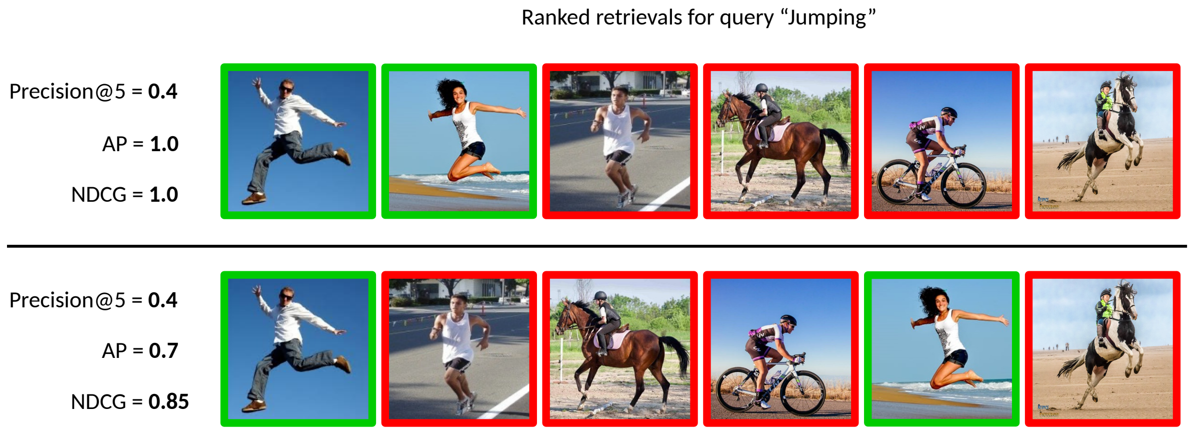


Figure 1.1: *Example demonstrating the relative sensitivity of evaluation measures to the ranks of the retrieved samples. While* AP *and* NDCG *are sensitive to the change in ranking, precision@k is not.*

example, Average Precision (AP) is a very well-known measure that is used for evaluating the performance of information retrieval systems. However, AP is not differentiable and also does not additively decompose onto individual samples. It is therefore difficult to optimize AP to train information retrieval systems. In this context, we look into the accuracy-of-approximation and difficulty-of-optimization based considerations involved in choice of loss functions, in the context of information retrieval systems.

Information retrieval systems require us to rank a set of samples according to their relevance to a query. Let us begin by providing a brief description of a general retrieval framework that employs a rank-based loss function, hereby referred to as the ranking framework. This framework is the same as or generalizes the ones that are popular in the literature [14, 45, 86, 105]. The risk of the predicted ranking is measured by a user-specified loss function. Several intuitive loss functions have been proposed in the literature. These include simple loss functions such as 0-1 loss [60, 70], loss based on evaluation measures like precision@k and area under the ROC curve (AUC) [46], as well as the more complex loss functions such as those based on average precision (AP) [13, 105], normalized discounted cumulative gain (NDCG) [14] and F1-score [46]. Some of these loss functions like the 0-1 loss and that based on precision@k are very weakly sensitive to the ranking order of the retrieval. For example, let us consider the example portrayed in Figure 1.1 which consists of two sets of ranked retrievals of images corresponding to the query "jumping". Assuming $+$ represents positive sample (green box in the figure) and $-$ a negative sample (red box) the two rankings can be represented as: $R_1 = [++----−]$ and $R_2 = [+---+−]$. In this case, precision@5 would have the same value of 0.4 for both R_1 and R_2. While intuitively it is quite clear that R_1 is a much better retrieval result compared to R_2, using precision@5 as an evaluation measure does not quite capture this difference in performance.

On the other hand, more complex evaluation measures like average precision have a stronger sensitivity to the ranking order of the retrieval. In the above illustrative example, the ranking R_1 would have an AP value of 1.0 compared to the lower AP value of 0.7 for ranking R_2. Similarly, the NDCG score for ranking R_1 is 1.0 compared to that of 0.85 for R_2. This is consistent with the intuition that R_1 is a more desirable retrieval ranking compared to R_2. Therefore, it makes more sense to use rank sensitive evaluation measures like AP and NDCG to measure the performance of the retrieval systems. Further, it would be desirable to use loss functions that are based on these evaluation measures to learn the model parameters of the retrieval system. However, these complex loss functions are generally difficult to optimize and can lead to highly inefficient training procedures. This difficulty in optimization often discourages usage of complex loss functions, like those based on AP and NDCG, for learning model parameters. Efficient optimization of such non-decomposable rank-based loss functions is a challenging problem that holds the key towards their application for learning information retrieval models.

1.3.3 Learning for Combinatorial Optimization

Combinatorial optimization problems are frequently encountered in the real world. They essentially include all problems that can be formulated as a function minimization or maximization over a finite discrete domain. Problems like the travelling salesman problem, scheduling problems and graph based problems like minimum spanning tree fall in this category. Most of the combinatorial optimization problems that are of interest in fields like artificial intelligence, machine learning and software engineering, are NP-hard. Exact algorithms like those based on exhaustive search or branch and bound techniques can not be scaled to reasonably big domains. For many problems, approximation algorithms that are often based on some type of relaxation of the original discrete problem can be designed. While designing such methods can be of theoretical interest and often provide worst case theoretical guarantees, their practical usage might be limited for large scale problems.

For many combinatorial optimization problems, heuristic algorithms are the most successful for finding a reasonably approximate solution in a feasible amount of time, particularly when looking at large scale problems. However, designing such heuristic algorithms generally requires expert knowledge of the specific problem domain and a certain amount of trial and error is involved in the process. This can be a tedious process to engage in and is difficult to repeat for every new problem. While meta-heuristic methods like simulated annealing and genetic algorithms provide a framework for designing heuristic algorithms for a wide class of problems, tailoring them to a specific problem again requires expert domain knowledge. Interestingly, most heuristics based combinatorial optimization algorithms can be thought of as sequential decision making processes. The decision in each iteration can be often modeled as a simple classification task that takes the current state of the system as input and selects an action from a discrete action space.

While exact and approximation algorithms work with a more rigid framework that allows for concrete theoretical analysis and guarantees, practical efficiency of these methods is again dependent on their capability to adapt to specific problem structures. For example, consider the branch and bound algorithm which searches for the optimal solution by partitioning the feasible space in a tree-structured manner. The partitioning of the feasible space is achieved by progressively splitting the space with respect to individual variables. In such a setting, the order in which the variables are considered for branching is critical for the efficiency of the algorithm. Unfortunately, there is no general method to come up with an optimal ordering, instead it comes down to designing good heuristic methods by leveraging the structure of the domain of interest. Here again lies the scope for using learning based methods to discover good heuristics from data instead of relying on domain experts to fulfill the onerous task of designing them every time we are dealing with a new domain. A similar idea is also relevant for approximation algorithms, whose efficiency can be improved with better heuristics that can potentially be learned from data.

A popular approach for devising approximate algorithms for combinatorial optimization problems is to relax the original discrete problem to a more tractable one. The most common approach is to relax the discrete domain to a continuous one. However, characterization of the relaxed continuous domain is non-trivial and might itself be computationally intractable. Therefore, there is often a trade off between the computational tractability and the tightness of the relaxation that one has to keep in mind. Moreover, solving the new relaxed problem would generally give a fractional solution that has to be rounded to get an integer solution. The rounding procedure is again non-trivial and the best performing ones are generally hand designed for specific problem domains. It should be possible to parameterize the space of relaxation and rounding procedures and apply learning based techniques to design approximation algorithms.

1.4 Contributions

Partial Linearization based Optimization for Multi-class SVM. Many tasks in computer vision can be formulated as multi-class classification problems. In other words, given an image or a video, the task is to assign it a label that belongs to a specified finite set. For example, in the case of object recognition from an image, the label can be car, chair or person. Similarly, for action recognition from a video, actions categories like jumping, kicking or clapping can be candidate labels. There has been extensive research in the area of multi-class classification with a plethora of solutions being proposed [19, 27, 64, 79]. We focused on multi-class SVM (MC-SVM), which is one of the most popular methods for this task. We proposed a novel partial linearization based approach for optimizing the multi-class SVM learning problem. Our method is an intuitive generalization of the Frank-Wolfe and the exponentiated gradient algorithms. In particular, it allows us to combine several of their desirable qualities into one approach: (i) the use of an expectation oracle (which provides the marginals over each output class) in order to estimate an informative descent direction, similar to exponentiated gradient; (ii) analytical computation of the optimal step-size in the descent direction that guarantees an increase in the dual objective, similar to Frank-Wolfe; and (iii) a block coordinate formulation similar to the one proposed for Frank-Wolfe, which allows for solving the problem for large datasets.

Efficient optimization of rank-based loss functions. The accuracy of information retrieval systems is often measured using complex loss functions such as the average precision (AP) or the normalized discounted cumulative gain (NDCG). Given a set of positive and negative samples, the parameters of a retrieval system can be estimated by minimizing these loss functions. However, the non-differentiability and non-decomposability of these loss functions does not allow for simple gradient based optimization algorithms. This issue is generally circumvented by either optimizing a structured hinge-loss upper bound to the loss function or by using asymptotic methods like the direct-loss minimization framework. Yet, the high computational complexity of loss-augmented inference, which is necessary for both the frameworks, prohibits its use in large training data sets. To alleviate this deficiency, we present a novel quicksort flavored algorithm for a large class of non-decomposable loss functions. Our algorithm has a superior runtime of $O(|\mathcal{N}| \log |\mathcal{P}| + |\mathcal{P}| \log |\mathcal{N}|)$ compared to $O(|\mathcal{P}| |\mathcal{N}| + |\mathcal{N}| \log |\mathcal{N}|)$ of [14, 105], where, $\mathcal{P}$ and $\mathcal{N}$ denote the sets of positive and negative samples respectively. We provide a complete characterization of the loss functions that are amenable to our algorithm, and show that it includes both AP and NDCG based loss functions. Furthermore, we prove that no comparison based algorithm can improve upon the computational complexity of our approach asymptotically. We also demonstrate that it is however possible to reduce the constant factors of the complexity by exploiting the special structure of specific loss functions like the AP loss. Apart from these exact methods, we additionally present an approach that approximates the AP loss over all samples by the AP loss over difficult samples (for example, those that are incorrectly classified by a binary SVM), while ensuring the correct classification of the remaining samples.

Learning to Round for Discrete Labeling Problems. Discrete labeling problems are often solved by formulating them as an integer program, and relaxing the integrality constraint to a continuous domain. While the continuous relaxation is closely related to the original integer program, its optimal solution is often fractional. Thus, the success of a relaxation depends crucially on the availability of an accurate rounding procedure. The problem of identifying an accurate rounding procedure has mainly been tackled in the theoretical computer science community through mathematical analysis of the worst-case. However, this approach is both onerous and ignores the distribution of the data encountered in practice. We present a novel interpretation of rounding procedures as sampling from a latent variable model, which opens the door to the use of powerful machine learning formulations in their design. Inspired by the recent success of deep latent variable models, we parameterize rounding procedures as a neural network, which lends itself to efficient optimization via back-propagation. By minimizing the expected value of the objective of the discrete labeling problem over training samples, we learn a rounding procedure that is more suited to the task at hand.

1.5 Outline of the Thesis

The rest of the thesis is organized in the following parts.

Efficient optimization for large scale classification. In this part, we discuss the problem of large scale classification where large number of output classes present an optimization challenge. We discuss our work based on the partial-linearization approach for efficient optimization of the multi-class SVM. This includes our work published in the European Conference on Computer Vision (ECCV) 2016.

Efficient optimization of rank-based loss functions. This part discusses the challenge posed by the complexity of some rank-based loss functions in terms of the difficulty of optimizing them for learning retrieval systems. We discuss our contribution towards characterizing a class of rank-based loss functions in terms of their amenability for efficient optimization and present our quick-sort flavoured algorithm for efficient optimization. This includes our work published in the Conference on Neural Information Processing Systems (NeurIPS) 2014 and the International Conference of Computer Vision and Pattern Recognition (CVPR) 2018.

Learning to optimize for combinatorial problems. In this part, we discuss the reverse problem of the potential use of machine leaning techniques for improving optimization, specifically combinatorial optimization. We discuss our contribution in developing a learning based framework for the rounding step of a continuous relaxation based approximation algorithm. This includes our work published in the International Conference on Artificial Intelligence and Statistics (AISTATS) 2018.

Conclusion and future work. This part of the thesis includes a summary of the contributions and draws directions for future research in related areas.

1.6 Publications

Part of the work described in this thesis has previously been presented as the following publications:

- **P. Mohapatra**, M. Rolinek, C. V. Jawahar, V. Kolmogorov and M. Pawan Kumar, *Efficient Optimization for Rank-based Loss Functions*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018. (Won best paper honorable mention award)

- **P. Mohapatra**, C. V. Jawahar and M. Pawan Kumar, *Learning to Round for Discrete Labeling Problems*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2018.

- **P. Mohapatra**, P. K. Dokania, C. V. Jawahar and M. Pawan Kumar, *Partial Linearization based Optimization for Multi-class SVM*, European Conference on Computer Vision (ECCV), 2016.

- A.Behl, **P. Mohapatra**, C. V. Jawahar and M. Pawan Kumar, *Optimizing Average Precision using Weakly Supervised Data*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2015.

- **P. Mohapatra**, C. V. Jawahar and M. Pawan Kumar, *Efficient Optimization for Average Precision SVM*, In Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2014.

Chapter 2

Partial Linearization based Optimization for Multi-class SVM

2.1 Introduction

Many tasks in computer vision can be formulated as multi-class classification problems. In other words, given an image or a video, the task is to assign it a label that belongs to a specified finite set. For example, in the case of object recognition from an image, the label can be car, chair or person. Similarly, for action recognition from a video, action categories like jumping, kicking or clapping can be candidate labels. There has been extensive research in the area of multi-class classification with a plethora of solutions being proposed [19, 27, 64, 79]. In this work, we focus on multi-class SVM (MC-SVM), which is one of the most popular methods for this task. The MC-SVM model provides a linear function that gives a score for each class. Given a test sample, its class is predicted by maximizing the score. During learning, the MC-SVM objective minimizes an upper bound on the empirical risk over the training data, for which we know the ground-truth labels. The risk is typically measured by the standard $0-1$ loss function. However, any other loss function can be easily substituted into the MC-SVM learning framework.

The size of the MC-SVM learning problem rapidly increases with the number of classes and size of the training dataset. In order to enable the use of MC-SVM with large scale problems, several optimization algorithms for minimizing its learning objective have been proposed in the literature. One of the most successful algorithms is a recent adaptation of the Frank-Wolfe algorithm [34]. Briefly, the algorithm solves the dual of the MC-SVM optimization problem iteratively. At each iteration, it obtains a descent direction by minimizing a linear approximation of the dual objective. It was shown in [44] that the computation of the descent direction corresponds to a call to the the so-called *max-oracle* for each sample. In other words, for each training sample, we maximize over the set of output classes with respect to the loss-augmented scores. As the max-oracle can be solved efficiently for the MC-SVM, the Frank-Wolfe algorithm can be effectively used to learn such models. There are two main advantages of the Frank-Wolfe algorithm. First, the optimal step-size in the descent direction can be computed analytically, thereby avoiding a tedious line search [44]. Second, it can be suitably modified to a block-coordinate version [43], where the max-oracle is solved for only one training sample at each iteration. The gain in efficiency obtained by this version does not affect the accuracy of the solution.

A key disadvantage of the Frank-Wolfe algorithm is that it only provides a very local approximation of the objective function with the aid of the max-oracle. In other words, it effectively focuses on one constraint (the most violated one) of the primal MC-SVM learning problem. In contrast, the exponentiated gradient algorithm [16] makes use of a more informative *expectation oracle*. To elaborate, instead of maximizing, it computes an expectation over the set of output classes with respect to a distribution parameterized by the loss-augmented scores. However, the exponentiated gradient algorithm suffers from the difficulty of choosing an optimal step-size, for which it has to resort to line search. Furthermore, despite the availability of a stochastic version of the algorithm, its worst-case time complexity is worse than that of the Frank-Wolfe algorithm.

Contributions. In this work, we propose a novel algorithm for optimizing the MC-SVM learning problem based on partial linearization [77]. Our algorithm provides a natural generalization to the Frank-Wolfe and the exponentiated gradient algorithms, thereby combining their desirable properties. Specifically, (i) it allows for the use of a potentially more informative descent direction based on the expectation oracle; (ii) it computes the optimal step-size in the descent direction analytically; and (iii) it can also be applied in a block coordinate fashion without losing the accuracy of the solution. We demonstrate the efficacy of our approach on the challenging computer vision problems of action classification, object recognition and gesture recognition using standard publicly available datasets. In certain cases, our method can also be used for efficient optimization of the more general structured SVM (SSVM) models. Specifically, in case of output spaces that have a low tree-width structure, we can employ efficient max-oracles and expectation-oracles. This in turn means that similar to the Frank-Wolfe [44] and exponentiated gradient algorithms [16], our method can be effectively used for learning an SSVM model. We demonstrate this on the problem of handwritten word recognition using a chain structured output space.

2.2 Related Work

Several algorithms have been proposed for optimizing multi-class SVMs. Most of the popular methods are iterative algorithms that make use of efficient sub-routines called *oracles* in each iteration [16, 44, 48, 81, 108]. The most popular algorithms can be bracketed into two classes depending on the type of oracles they use: ones that use the max-oracle [44, 48, 81]; and the ones that use the expectation oracle [16, 108].

A max-oracle sub-routine maximizes the loss-augmented score over the output space. In other words, given the current estimate of the parameters and a training sample, it returns the output that maximizes the sum of the classifier score and the loss. The sub-gradient descent algorithm [81] calls the max-oracle to compute the sub-gradient of the primal objective and uses it as the update direction in each iteration. The cutting-plane algorithm [48] uses the max-oracle to get the most violating constraint or the cutting plane. It accumulates the cutting planes to generate increasingly accurate approximations to the primal problem that it solves in each iteration. The recent adaptation [44] of the Frank-Wolfe algorithm to the MC-SVM and SSVM learning problems uses the max-oracle to compute the conditional gradient of the dual problem. All three aforementioned algorithms have a complexity of $O(1/\epsilon)$, where ϵ is the user-specified optimization tolerance. However, in practice, the block-coordinate Frank-Wolfe algorithm has been shown to provide faster convergence on a variety of problems [44].

In contrast to the max-oracle, the expectation-oracle computes an expectation over the output space with respect to a distribution parameterized by the loss-augmented scores. In [16], the expectation-oracle is used to make exponentiated gradient updates [54], which guarantees descent in each iteration. The Bregman projection based excessive gap reduction technique presented in [108] also uses the expectation oracle. While this algorithm has a highly competitive complexity of $O(1/\sqrt{\epsilon})$, the method does not work with noisy oracles and hence cannot lend itself to a stochastic or a block-coordinate version.

As will be seen shortly, our approach naturally generalizes over algorithms from both the categories with the use of a temperature hyperparameter. When the temperature is set to 0, the expectation oracle resembles the max-oracle and our method reduces to the Frank-Wolfe algorithm[44]. Importantly, for a non-zero temperature, the use of the expectation-oracle can provide us with a less local approximation of the objective function. Hence, for the multi-class SVM learning problem, it may be beneficial to use the expectation-oracle instead of the max-oracle. Another key aspect of our algorithm is that it chooses an optimal step-size at each iteration. If we instead fix the step-size to 1 in every iteration and use a non-zero value of the temperature hyperparameter, then our method reduces to the exponentiated gradient algorithm [16]. Moreover, unlike the cutting plane [48] and the excessive gap reduction [108] algorithms our approach allows for a block-coordinate version, which leads to faster rate of convergence without affecting the accuracy of the solution.

2.3 Preliminaries

2.3.1 The Multi-class SVM Optimization Problem

We provide a brief overview of the multi-class SVM (MC-SVM) formulated as a structured SVM learning problem [94]. Given an input $\mathbf{x} \in \mathcal{X}$ the aim of multi-class classification is to predict the output y that belongs to the output space $\mathcal{Y}$. If the number of classes is denoted by c, the output space $\mathcal{Y} = \{1, \ldots, c\}$. Let the feature representation of sample $\mathbf{x}$ be $\boldsymbol{\varphi}(\mathbf{x})$, then a joint feature map $\Phi(\mathbf{x}, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}^d$ is defined as

$$\Phi(\mathbf{x}, y) = [\boldsymbol{v}_1^\top \ \ldots \ \boldsymbol{v}_j^\top \ \ldots \ \boldsymbol{v}_c^\top]^\top \tag{2.1}$$
$$where,\ \boldsymbol{v}_j = \begin{cases} \boldsymbol{\varphi}(\mathbf{x}) & \text{if } j = y, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

A multi-class SVM, parameterized by $\mathbf{w}$, provides a linear prediction rule as follows:

$$h_{\mathbf{w}}(\mathbf{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \left(\mathbf{w}^\top \Phi(\mathbf{x}, y) \right).$$

Given a set of labelled samples $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, the parameter vector $\mathbf{w}$ is learnt by solving the following convex optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \mathbf{w}^\top \Psi_i(y) \geq \Delta(y_i, y) - \xi_i, \forall i \in [n], \forall y \in \mathcal{Y} \end{aligned} \tag{2.2}$$

Here, $\Psi_i(y) = \Phi(\mathbf{x}_i, y_i) - \Phi(\mathbf{x}_i, y)$ and the loss incurred for predicting y, given the ground truth y_i for the sample $\mathbf{x}_i$, is defined as

$$\Delta(y_i, y) = \begin{cases} 0 & \text{if } y = y_i, \\ 1 & \text{otherwise.} \end{cases} \tag{2.3}$$

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and shall use $\Delta_i(y)$ as a short hand for $\Delta(y, y_i)$. The Lagrangian dual of problem (2.2) is given by:

$$\begin{aligned} \min_{\boldsymbol{\alpha} \geq 0} \quad & T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_{y \in \mathcal{Y}} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n]. \end{aligned} \tag{2.4}$$

Here the dual variable vector $\boldsymbol{\alpha}$ is of size $m = n \times c$; $\mathbf{b} \in \mathcal{R}^m$ is defined as $\mathbf{b} = \{b_{iy} = \frac{1}{n}\Delta_i(y) \mid i \in [n], y \in \mathcal{Y}\}$ and the matrix $A \in \mathcal{R}^{d \times m}$ is defined as $A = \{A_{iy} = \frac{1}{\lambda n}\Psi_i(y) \in \mathcal{R}^d \,|\, i \in [n], y \in \mathcal{Y}\}$.

It is possible to cheaply evaluate the objective of the primal MC-SVM formulation since the following problem lends itself to efficient optimization. Specifically, in order to compute the MC-SVM objective at a given set of parameters $\mathbf{w}$, we can solve the following problem for each sample i.

$$\bar{y}_i = \operatorname*{argmax}_{y \in \mathcal{Y}} \Delta_i(y) - \mathbf{w}^\top \Psi_i(y). \tag{2.5}$$

Given $\overline{y}_i$, the value of the slack variable $\xi_i = \Delta_i(\overline{y}_i) - \mathbf{w}^\top \Psi_i(\overline{y}_i)$. We refer to the above problem as the *max-oracle*. Let $P(y)$ denote the probability distribution over the set of output classes, parameterized by the loss augmented scores, that is,

$$P(y) = \frac{\exp\big(\Delta_i(y) - \mathbf{w}^\top \Psi_i(y)\big)}{\sum_{y \in \mathcal{Y}} \exp\left(\Delta_i(y) - \mathbf{w}^\top \Psi_i(y)\right)}. \tag{2.6}$$

The max-oracle gives the most probable class according to the distribution $P(y)$. It has been shown through several works, including cutting-plane algorithms [48], subgradient descent [81] and Frank-Wolfe [43], that an inexpensive max-oracle is sufficient to minimize problem (2.2) and/or its Lagrangian dual (2.4) efficiently.

As we will see shortly, our work exploits the fact that, for multi-class classification problems, a related problem known as the *expectation-oracle* can be solved efficiently as well (with the same time complexity as the max-oracle). While the max-oracle gives the most probable class, the expectation-oracle returns an expectation over the complete output space with respect to the distribution $P(y)$. By cleverly exploiting this observation, we obtain a natural generalization of the Frank-Wolfe algorithm that retains many of its desirable properties such as: guaranteed descent direction, analytically computable optimal step size and guaranteed convergence even in block-coordinate mode. At the same time it also allows the use of the expectation-oracle to find a valid descent direction that can often lead to improved performance in practice.

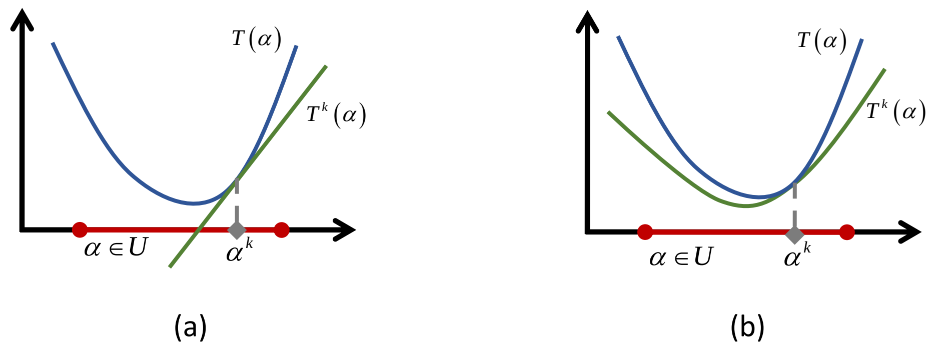


Figure 2.1: *Illustration of linear vs. partially-linear local approximation of a function: (a) Frank-Wolfe solves a linear local approximation of the function, whereas, (b) Partial linearization based approach solves a local convex approximation of the function, over the compact domain.*

2.3.2 Partial Linearization

Let us consider the following optimization problem with a convex and continuously differentiable objective $T(\boldsymbol{\alpha})$ defined over a compact and convex feasible set U: $\min_{\boldsymbol{\alpha}\in U} \; T(\boldsymbol{\alpha})$. For this problem, Patriksson [77] proposes a framework that unifies several feasible-direction finding methods for non-linear optimization through the concept of partial linearization of the objective function. The idea of partial linearization is to construct a convex approximation to the original objective $T(\boldsymbol{\alpha})$ at each iteration. The approximation involves switching the original function with a substitute function. Furthermore, in order to model the difference between the original function and the substitute function, we add a first order approximation of this difference. Figure 2.1 illustrates the local approximation made by the partial linearization based method compared to a linear approximation as made in Frank-Wolfe [44].

Formally, at each iteration k, we solve the following problem:

$$\begin{aligned}\min_{\boldsymbol{\alpha}\in U} \; T^k(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha},\boldsymbol{\alpha}^k) + T(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^k,\boldsymbol{\alpha}^k) \\ + [\nabla T(\boldsymbol{\alpha}^k) - \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^k,\boldsymbol{\alpha}^k)]^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^k).\end{aligned} \tag{2.7}$$

The term $f(\boldsymbol{\alpha},\boldsymbol{\alpha}^k)$ denotes the substitute function defined at the current solution $\boldsymbol{\alpha}^k$. The term $T(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^k,\boldsymbol{\alpha}^k) + [\nabla T(\boldsymbol{\alpha}^k) - \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^k,\boldsymbol{\alpha}^k)]^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^k)$ is the first order Taylor expansion of the actual error term $T(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha},\boldsymbol{\alpha}^k)$ and is used as an approximation for it. Patriksson [77] showed that the approximation proposed in equation (2.7) actually preserves the gradient of the original objective function.

This guarantees that a valid descent direction for the approximate problem (2.7) is also a valid descent direction for the original problem. The optimal solution $\bar{\boldsymbol{\alpha}}^k$ to problem 2.7 gives a descent direction. This allows us to update the solution as $\boldsymbol{\alpha}^{k+1} = (1-\gamma)(\boldsymbol{\alpha}^k) + (\gamma)(\bar{\boldsymbol{\alpha}}^k)$, where γ is the step-size that can be determined via line search in general. Interestingly, in some special cases, including the one considered in this work, the optimal step-size can also be computed analytically, which avoids the tedious line search. For convergence, $f(\mathbf{x}, \mathbf{y})$ has to be convex and continuously differentiable with respect to $\mathbf{x}$ and continuous with respect to $\mathbf{y}$. We adapt the partial linearization method for solving problem (2.4) in the following section.

2.4 Partial Linearization for Multi-class SVM Optimization

The dual multi-class SVM problem defined in problem (2.4) has a compact convex feasible set and has a continuously differentiable convex objective. This allows us to use the partial linearization method to solve the optimization problem. However, as the above description shows, partial linearization is a very general framework. For it to be applied successfully, we need to ensure that we make the right choice for the substitute function. Specifically, the resulting problem (2.7) must lend itself to efficient optimization. Furthermore, in our case, we would like to ensure that problem (2.7) captures the information regarding how much each constraint of the primal multi-class SVM problem is violated, similar to the expectation-oracle. To this end, we define the substitute function as follows:

$$f(\boldsymbol{\alpha}, \boldsymbol{\alpha}^k) = \frac{\tau}{n} \sum_{i \in [n]} \sum_{y \in \mathcal{Y}} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}). \tag{2.8}$$

Here, τ is a non-negative hyperparameter, which we refer to as the temperature. Here, the function $f(\alpha, \alpha^k)$ is independent of α^k and is similar to a mirror function as employed in a mirror descent method. In the following subsection, we show that for the above choice of substitute function, the partial linearization approach generalizes both the Frank-Wolfe and the exponentiated gradient algorithm.

2.4.1 Partial linearization in the dual space

When we use the substitute function defined in equation (2.8) for partial linearization of the dual multi-class SVM problem, the form of the update direction in the resulting optimization algorithm is described by the following proposition.

Proposition 1. *If the substitute function is defined as*

$$f(\boldsymbol{\alpha}, \boldsymbol{\alpha}^k) = \frac{\tau}{n} \sum_{i \in [n]} \sum_{y \in \mathcal{Y}} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}),$$

then the update direction $\mathbf{s}^k$ *in iteration* k *for given* i *and* $y \in \mathcal{Y}$ *can be computed as*

$$\mathbf{s}_{iy}^k = \frac{\exp\left(\log(\boldsymbol{\alpha}_{iy}^{k-1}) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k-1^\top}\Psi_i(y))\right)}{z_i}. \tag{2.9}$$

Proof. Given the choice of the substitute function as,

$$f(\boldsymbol{\alpha}, \boldsymbol{\alpha}^k) = \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}), \tag{2.10}$$

the objective of the optimization problem that has to be solved in the k^{th} iteration becomes,

$$\begin{aligned} T^k(\boldsymbol{\alpha}) \quad &= \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} \log(\boldsymbol{\alpha}_{iy}) - b^\top \boldsymbol{\alpha}^k \\ &+ \frac{\lambda}{2} \boldsymbol{\alpha}^{k^\top} A^\top A \boldsymbol{\alpha}^k \\ &- \frac{\tau}{n} \sum_{i \in [n]} \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy}^k \log(\boldsymbol{\alpha}_{iy}^k) \\ &+ ((-b + \lambda A^\top A \boldsymbol{\alpha}^k) - \frac{\tau}{n}(1 + \log(\boldsymbol{\alpha}^k)))^\top (\boldsymbol{\alpha} - \boldsymbol{\alpha}^k) \end{aligned}$$

Therefore, the approximate optimization problem to be solved in the k^{th} iteration looks like,

$$\begin{aligned} \min_{\alpha \geq 0} \quad & T^k(\boldsymbol{\alpha}) \\ \text{s.t.} \quad & \sum_{y \in Y_i} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n] \end{aligned} \tag{2.11}$$

The Lagrangian of problem 2.11, with $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_n]^\top$ as the set of Lagranian multipliers can be defined as,

$$L(\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}) = T^k(\boldsymbol{\alpha}) + \sum_{i \in [n]} \boldsymbol{\beta}_i \left(\sum_{y \in Y_i} \boldsymbol{\alpha}_i - 1 \right)$$

Differentiating the Lagrangian with respect to the variables,

$$\begin{aligned}
& \frac{\partial L}{\partial \boldsymbol{\alpha}_{iy}} = 0 \\
\Rightarrow \quad & \frac{\tau}{n}(1+\log(\boldsymbol{\alpha}_{iy})) + (-b_{iy} + \lambda (A^T A)_{iy} \boldsymbol{\alpha}^k) \\
& -\frac{\tau}{n}(1+\log(\boldsymbol{\alpha}_{iy}^k)) + \boldsymbol{\beta}_i = 0 \\
\Rightarrow \quad & \frac{\tau}{n}(1+\log(\boldsymbol{\alpha}_{iy})) + (-b_{iy} + \frac{1}{n}\mathbf{w}^{k\top}\Psi_i(\mathbf{y})) \\
& -\frac{\tau}{n}(1+\log(\boldsymbol{\alpha}_{iy}^k)) + \boldsymbol{\beta}_i = 0 \\
\Rightarrow \quad & \frac{\tau}{n}\log(\boldsymbol{\alpha}_{iy}) = \frac{\tau}{n}\log(\boldsymbol{\alpha}_{iy}^k) + b_{iy} - \frac{1}{n}\mathbf{w}^{k\top}\Psi_i(\mathbf{y}) - \boldsymbol{\beta}_i \\
\Rightarrow \quad & \boldsymbol{\alpha}_{iy} = \exp\left(\log(\boldsymbol{\alpha}_{iy}^k) + \frac{n}{\tau}(b_{iy} - \frac{1}{n}\mathbf{w}^{k\top}\Psi_i(\mathbf{y}) - \boldsymbol{\beta}_i)\right) \\
\Rightarrow \quad & \boldsymbol{\alpha}_{iy} = \frac{\exp\left(\log(\alpha_{iy}^k) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k\top}\Psi_i(\mathbf{y})\right)}{\exp\left(\frac{n}{\tau}\beta_i\right)} \qquad (2.12)
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial L}{\partial \boldsymbol{\beta}_i} = 1 \;\Rightarrow\; \sum_{y \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} = 0 \\
\Rightarrow \quad & \sum_{y \in \mathcal{Y}_i} \frac{\exp\left(\log(\boldsymbol{\alpha}_{iy}^k) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k\top}\Psi_i(\mathbf{y}))\right)}{\exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right)} = 1 \\
\Rightarrow \quad & \sum_{y \in \mathcal{Y}_i} \exp\left(\log(\boldsymbol{\alpha}_{iy}^k) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k\top}\Psi_i(\mathbf{y}))\right) = \exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right) \\
\Rightarrow \quad & z_i = \exp\left(\frac{n}{\tau}\boldsymbol{\beta}_i\right) \qquad (2.13)
\end{aligned}$$

Hence, using 2.12 and 2.13, the optimal solution is,

$$\mathbf{s}_{iy}^k = \frac{\exp\left(\log(\alpha_{iy}^k) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k\top}\Psi_i(\mathbf{y}))\right)}{z_i}$$

□

In equation 2.9, for each sample i, $\mathbf{s}_i^k(y)$ forms a probability distribution over the set of classes $\mathcal{Y}$. In the primal setting, this is equivalent to having an expectation over the entire output space as the update direction and is therefore similar to an expectation-oracle. In each iteration of the algorithm, given the update direction, we need to perform a line search to find the optimal step size γ in that direction. Since the dual multi-class SVM problem involves optimizing a quadratic function, it is possible to analytically compute the optimal step-size. The following proposition that gives the form of the optimal step size directly follows from the work of Jaggi *et al.* [43].

Proposition 2. *The optimal step-size along the update direction* $\mathbf{s}^k$ *can be computed to be equal to*

$$\gamma = \frac{< \boldsymbol{\alpha}^{k-1} - \mathbf{s}^k, -\mathbf{b} + A^\top A\boldsymbol{\alpha}^{k-1} >}{\lambda||A(\boldsymbol{\alpha}^{k-1} - \mathbf{s}^k)||^2}. \tag{2.14}$$

Here it should be observed that setting the temperature parameter τ to 0 results in a distribution $\mathbf{s}_i^k$ that has probability 1 for the label

$$\bar{y}_i = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \left(L_i(y) - \mathbf{w}^{k-1^T} \Psi_i(y) \right)$$

and 0 elsewhere. This results in an update direction that is the same as that of the Frank-Wolfe algorithm and thus reduces the partial linearization method to the Frank-Wolfe algorithm [43]. Moreover, fixing the step-size γ to 1 for all iterations reduces our approach to the exponentiated-gradient algorithm [16]. Hence, our partial linearization based approach for optimizing the MC-SVM problem generalizes both the Frank-Wolfe as well as the exponentiated-gradient algorithms. Importantly, the descent direction obtained using some $\tau > 0$ can be significantly better than that obtained using $\tau = 0$. This is illustrated in the following example.

In Problem 2.4, let $n = 1$, $\lambda = 1$, $A = [2, 0, 0; 0, 1, 0; 0, 0, 3]$ and $b = [1; 1; 0]$. Assume, after the $(k-1)^{th}$ iteration of the optimization algorithm, the location in the feasible set is $\alpha^{k-1} = [0.125, 0.5, 0.5]^\top$. Now, if we take $\tau = 0$, the descent direction for the k^{th} iteration can be computed to be $\mathbf{s}^k_{\tau=0} = [1, 0, 0]^\top$. Similarly, for $\tau = 1$, the descent direction would be $\mathbf{s}^k_{\tau=1} = [0.199, 0.796, 0.005]^\top$. In each case, we take the optimal step in the descent direction. It can be verified that while the step along $s^k_{\tau=0}$ reduces the objective function by 0.5341, the step along $s^k_{\tau=1}$ reduces the objective function by a bigger value of 1.2550. This is primarily due to the fact that the Frank-Wolfe algorithm ($\tau = 0$) constraints the descent directions to be only towards vertices of the feasible domain polytope. For instance in this example, $\mathbf{s}^k_{\tau=0}$ can only take values from among $\{[1, 0, 0]^\top, [0, 1, 0]^\top, [0, 0, 1]^\top\}$, which prevents it from taking a more direct path towards the optimal solution ($[0.25, 1, 0]^\top$) which lies on one of the facets of the polytope and hence away from the direction of any of the vertices. On the other hand, with $\tau > 0$, our algorithm can explore more direct descent paths towards the solution.

The partial linearization algorithm for optimizing the dual MC-SVM problem is outlined in Algorithm 1. Step 6 in Algorithm 1 requires us to explicitly compute the update direction corresponding to each dual variable. For the MC-SVM problem, as the number of dual variables is a reasonable $(number\ of\ samples) \times (number\ of\ classes)$, $\mathbf{s}^k_{iy}$ can be efficiently computed for every sample $\mathbf{x}_i$ as the marginal probability of each class y. Once we have the update direction we take a step in that

Algorithm 1 *Partial linearization for optimizing multi-class* SVM

```
1:  $\mathcal{D} = (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_n, y_n)$
2:  Initialize $\boldsymbol{\alpha}^0$ such that $\mathbf{w}(\boldsymbol{\alpha}^0) \sim [0]^d$, $k \leftarrow 1$
3:  **repeat**
4:      **for** $i \in [n]$ **do**
5:          $\forall y \in \mathcal{Y}$,
6:          $\mathbf{s}_{iy}^k \leftarrow \frac{\exp\left(\log(\boldsymbol{\alpha}_{iy}^{k-1}) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k-1^\top}\Psi_i(y))\right)}{z_i}$
7:      Optimal step size, $\gamma \leftarrow \frac{<\boldsymbol{\alpha}^{k-1} - \mathbf{s}^k, -\mathbf{b} + A^\top A\boldsymbol{\alpha}^{k-1}>}{\lambda||A(\boldsymbol{\alpha}^{k-1} - \mathbf{s}^k)||^2}$
8:      Update $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}^k \leftarrow (1-\gamma)\boldsymbol{\alpha}^{k-1} + (\gamma)\mathbf{s}^k$
9:      Update $\mathbf{w}$, $\mathbf{w}^k \leftarrow A\boldsymbol{\alpha}^k$
10:     $k \leftarrow k + 1$
11: **until** Convergence
12: Optimal parameter, $\mathbf{w}$
```

direction with optimal step-size γ as computed in Step 8. Then the dual and the primal variables are updated to complete an iteration of the algorithm.

2.4.2 Block-Coordinate Partial Linearization

In many tasks, it is very common to learn classification models using very large datasets. In such scenarios, learning an MC-SVM model using the partial linearization algorithm described in Algorithm 1 can be very slow. This is because, each update iteration of Algorithm 1 requires a pass through the entire dataset. In order to circumvent this expensive step, we present a block-coordinate version of the algorithm, which updates the model parameters after every single sample encounter. Algorithm 2 outlines the details of the block-coordinate partial linearization algorithm. The key difference is that, unlike Algorithm 1, Algorithm 2 does not have to loop through all the samples in the training set before updating the primal variable vector. Instead, we pick a random sample i from the training set (step 5) and compute the marginals just for this sample. Accordingly we update the primal weight vector $\mathbf{w}$ with this new marginal for sample i while the marginals for all other samples remain unchanged. This is similar to the coordinate descent method and is more efficient compared to the batch method as instead of solving n convex optimization problems, we have to solve only one in each iteration. As shown in the following proposition, this improvement in run-time does not affect the accuracy of the solution.

Algorithm 2 *Block-Coordinate Partial linearization for optimizing multi-class* SVM

```
1: $\mathcal{D} = (\mathbf{x}_i, y_i), \dots, (\mathbf{x}_n, y_n)$
2: Initialize $\boldsymbol{\alpha}^0$ such that $\mathbf{w}(\boldsymbol{\alpha}^0) \sim [0]^d$, $k \leftarrow 1$
3: Initialize a $(d \times n)$ matrix $W$ such that $i^{th}$ column of $W$, $\mathbf{w}_i = \mathbf{w}(\boldsymbol{\alpha}_i^0)$
4: repeat
5:     Chose a random $i \in [n]$
6:     $\forall y \in \mathcal{Y}$,
7:     $\mathbf{s}_{iy}^k \leftarrow \frac{\exp\left(\log(\boldsymbol{\alpha}_{iy}^{k-1}) + \frac{1}{\tau}(\Delta_i(y) - \mathbf{w}^{k-1^\top}\Psi_i(y))\right)}{z_i}$
8:     Optimal step size, $\gamma \leftarrow \frac{<\boldsymbol{\alpha}_i^{k-1} - \mathbf{s}_i^k, -\mathbf{b} + A^\top A \boldsymbol{\alpha}_i^{k-1}>}{\lambda ||A(\boldsymbol{\alpha}_i^{k-1} - \mathbf{s}_i^k)||^2}$
9:     Update $\boldsymbol{\alpha}_i$, $\boldsymbol{\alpha}_i^k \leftarrow (1-\gamma)\boldsymbol{\alpha}_i^{k-1} + (\gamma)\mathbf{s}_i^k$
10:    Update $\mathbf{w}_i$, $\mathbf{w}_i^k \leftarrow A\boldsymbol{\alpha}_i^k$
11:    Update $\mathbf{w}$, $\mathbf{w}^k \leftarrow \mathbf{w}^{k-1} - \mathbf{w}_i^{k-1} + \mathbf{w}_i^k$
12:    $k \leftarrow k + 1$
13: until Convergence
14: Optimal parameter, $\mathbf{w}$
```

Proposition 3. *The block-coordinate partial linearization algorithm is guaranteed to converge to the global optima of the multi-class* SVM *learning problem.*

Proof. Consider the following optimization problem whose feasible domain and objective function are a cross-section of the original dual SSVM learning problem.

$$\begin{aligned} \min_{\boldsymbol{\alpha}_k \geq 0} \quad & T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_{\mathbf{y} \in \mathcal{Y}_j} \boldsymbol{\alpha}_{jy} = 1, \\ & \boldsymbol{\alpha}_i = \bar{\boldsymbol{\alpha}}_i, \forall i \in [n] - j, \end{aligned} \tag{2.15}$$

where $\bar{\boldsymbol{\alpha}}_i$'s are fixed and satisfy $\sum_{\mathbf{y} \in \mathcal{Y}_i} \bar{\boldsymbol{\alpha}}_{iy} = 1, \forall i \in [n] - j$. We can devise a partial-linearization algorithm for solving this optimization problem using $f(\boldsymbol{\alpha}_j, {\boldsymbol{\alpha}_j}^k) = \frac{\tau}{n}\sum_{\mathbf{y} \in \mathcal{Y}_j} \boldsymbol{\alpha}_{jy} \log(\boldsymbol{\alpha}_{jy})$, as the substitute function. Following the procedure similar to the proof for proposition 1, it can be shown that the update direction in the k^{th} iteration of the algorithm would take the following form.

$$s_{iy}^k = \frac{\exp\left(\log(\alpha_{jy}^{k-1}) + \frac{1}{\tau}(\Delta_j(y) - \mathbf{w}^{k-1^T}\Psi_{jy})\right)}{z_j}. \tag{2.16}$$

It follows from the work of Patriksson in [77] that this update direction is guaranteed to be a feasible descent direction of the objective function of problem 2.15. As the objective function in 2.15 is a cross-section of the objective function of problem (2) over $\boldsymbol{\alpha}_j$, this update direction would also be a feasible descent direction of the original objective of the dual SSVM learning problem. It can be easily verified that in the k^{th} iteration of the block-coordinate partial-linearization (BCPL) algorithm, the update direction is of the form of 2.16 with $\bar{\boldsymbol{\alpha}}_{iy} = \boldsymbol{\alpha}_{iy}^k, \forall i \in [n] - j$. Therefore, the update direction in each iteration of BCPL is guaranteed to reduce the original dual SSVM objective. Since the dual SSVM learning problem is convex, the BCPL algorithm is guaranteed to converge to the global optima.

$\square$

2.5 Partial Linearization for Structured SVM Optimization

The multi-class SVM solves a prediction problem in which the output space is a set of classes. However, for many tasks, the output space can have a more complicated structure. The structured SVM (SSVM), which is a generalization of the binary SVM to structured output spaces, can effectively model such structures. Given an input $\mathbf{x} \in \mathcal{X}$, the aim is to predict the output $\mathbf{y}$ that belongs to a structured space $\mathcal{Y}(\mathbf{x})$. Borrowing the notations from section 2.3.1, a structured SVM, parameterized by $\mathbf{w}$, provides a linear prediction rule as follows: $h_{\mathbf{w}}(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} \left(\mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y})\right)$. Given a set of labelled samples $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, the parameter vector $\mathbf{w}$ is learnt by solving the following convex optimization problem:

$$\min_{\mathbf{w},\xi} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i, \quad \text{s.t. } \mathbf{w}^\top \Psi_i(\mathbf{y}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i, \forall i \in [n], \forall \mathbf{y} \in \mathcal{Y}_i \tag{2.17}$$

The key differences from the multi-class SVM formulation are that here we can have any general joint feature map $\Phi(\mathbf{x}, \mathbf{y})$ and loss function $\Delta(\mathbf{y}_i, \mathbf{y})$ designed to effectively model the structure of the output space. The Lagrangian dual of problem (2.19) is given by:

$$\min_{\boldsymbol{\alpha}\geq 0} \qquad T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha}, \qquad \text{s.t. } \sum_{\mathbf{y}\in\mathcal{Y}_i} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n]. \tag{2.18}$$

Here the dual variable vector $\boldsymbol{\alpha}$ is of size $m = \sum_{i=1}^{n} |\mathcal{Y}_i|$; $\mathbf{b}$ and A have the same definition as in section 2.3.1.

In general the size of output space can be exponential in the number of output variables. This would result in exponentially large number of primal constraints and dual variables, which can be hard to deal with. However, these problems can be overcome by making clever use of the structure of the output space. The key observation behind our effective partial linearization based optimization algorithm is that we can efficiently compute the marginals of the output variables. Now, when the output space $\mathcal{Y}_i$ has a low tree-width graph structure, it is possible to efficiently compute the exact marginals of the output variables, by solving the expectation-oracle problem. This can be done using a message passing algorithm over a junction tree corresponding to the underlying graph of the output space [100]. In such a setting, our algorithm can be used for efficient optimization of the SSVM learning problem for low tree-width models. We discuss the partial-linearization algorithm for learning low tree-width SSVM models in detail in the following subsection.

2.5.1 Partial linearization with primal update for optimization of low tree-width SSVM

We provide a brief overview of the structured SVM optimization problem. Given an input $\mathbf{x} \in \mathcal{X}$ the aim of structured prediction is to predict the output $\mathbf{y}$ that belongs to a structured space $\mathcal{Y}(\mathbf{x})$. A joint feature map $\Phi(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}^d$, encodes the relationship between an input $\mathbf{x}$ and an output $\mathbf{y}$. A structured SVM, parameterized by $\mathbf{w}$, provides a linear prediction rule as follows: $h_{\mathbf{w}}(\mathbf{x}) = \text{argmax}_{\mathbf{y} \in \mathcal{Y}} \left(\mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y})\right)$, which is parameterized by $\mathbf{w}$. Given a set of labelled samples $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, the parameter vector $\mathbf{w}$ is learnt by solving the following convex optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \mathbf{w}^\top \Psi_i(\mathbf{y}) \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i, \forall i \in [n], \forall \mathbf{y} \in \mathcal{Y}(\mathbf{x}_i) \end{aligned} \tag{2.19}$$

Here $\Psi_i(\mathbf{y}) = \Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \mathbf{y})$ and $\Delta(\mathbf{y}_i, \mathbf{y})$ is the loss incurred for predicting $\mathbf{y}$ given the ground truth $\mathbf{y}_i$ for the sample $\mathbf{x}_i$. We use $[n]$ to denote the set $\{1, 2, \dots, n\}$ and shall use $\mathcal{Y}_i$ and $\Delta_i(\mathbf{y})$ as a short hand for $\mathcal{Y}(\mathbf{x}_i)$and $\Delta(\mathbf{y}, \mathbf{y}_i)$ respectively. The Lagrangian dual of the problem (2.19) is given by:

$$\begin{aligned} \min_{\boldsymbol{\alpha} \geq 0} \quad & T(\boldsymbol{\alpha}) = -\mathbf{b}^\top \boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^\top A^\top A \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_{\mathbf{y} \in \mathcal{Y}_i} \boldsymbol{\alpha}_{iy} = 1, \forall i \in [n]. \end{aligned} \tag{2.20}$$

Here the dual variable vector $\boldsymbol{\alpha}$ is of size $m = \sum_{i=1}^{n} |\mathcal{Y}_i|$; $\mathbf{b} \in \mathcal{R}^m$ is defined as $\mathbf{b} = \{b_{iy} = \frac{1}{n}\Delta_i(\mathbf{y}) \mid i \in [n], \mathbf{y} \in \mathcal{Y}_i\}$ and the matrix $A \in \mathcal{R}^{d \times m}$ is defined as

$$A = \{A_{iy} = \frac{1}{\lambda n}\Psi_i(\mathbf{y}) \in \mathcal{R}^d \,|\, i \in [n], \mathbf{y} \in \mathcal{Y}_i\}.$$

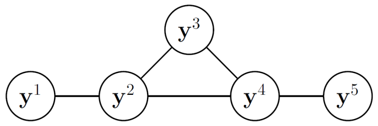


Figure 2.2: *An example of a plausible underlying graph structure of the output space. Here, the graph has 3 maximal cliques:* $\mathcal{C}_1 = \{\mathbf{y}^1, \mathbf{y}^2\}$, $\mathcal{C}_2 = \{\mathbf{y}^2, \mathbf{y}^3, \mathbf{y}^4\}$, $\mathcal{C}_3 = \{\mathbf{y}^4, \mathbf{y}^5\}$

.

In general the size of output space can be exponential in the number of output variables. This would result in exponentially large number of primal constraints and dual variables which can be hard to deal with. For example, consider the problem of recognizing the handwritten word depicted in an image, which has been segmented into p letters. The total number of possible words is 26^p. In other words, given the current set of parameters, we would require $O(n26^p)$ time in order to compute the objective function of Problem (2.19), since we would have to find the value of the slack variable ξ_i for each sample. In order to alleviate this deficiency, the structure of the output space is often exploited. For example, in the handwritten word recognition example, we can consider the output space to consist of a set of p parts. Each part corresponds to a letter of the word. Consider a joint feature vector $\Psi_i(\mathbf{y})$ that decomposes over the parts, that is, $\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1); \Psi_i(\mathbf{y}^2); ...; \Psi_i(\mathbf{y}^p)]$. It can be verified that, for a given set of parameters, we can compute the objective function value of Problem (2.19) in $O(np)$ time. However, the above joint feature vector fails to capture the interdependency between the letters. For example, if the q^{th} letter is a 'q' then the likelihood of the $(q+1)^{th}$ letter to be 'u' is very high. To capture this, one can use a slightly more complex joint feature vector of the form:

$$\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1, \mathbf{y}^2); ...; \Psi_i(\mathbf{y}^q, \mathbf{y}^{q+1}); ...; \Psi_i(\mathbf{y}^{p-1}, \mathbf{y}^p)].$$

More generally, one can visually represent the output space as a graph where the vertices are the parts of the output and the edges represent their interdependency. Using a "Markovian" style argument, we define a joint feature vector that decomposes into features defined over the maximal cliques of the graph. In other words, $\Psi_i(\mathbf{y}) = [\Psi_i(\mathcal{C}_1); \Psi_i(\mathcal{C}_2); ...; \Psi_i(\mathcal{C}_h)]$, where $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_h\}$ is the set of maximal cliques of the graph. For example, consider the graph shown in Figure 2.2. In this case, the graph consists of 3 maximal cliques and the joint feature vector decomposes as:

$$\Psi_i(\mathbf{y}) = [\Psi_i(\mathbf{y}^1, \mathbf{y}^2); \Psi_i(\mathbf{y}^2, \mathbf{y}^3, \mathbf{y}^4); \Psi_i(\mathbf{y}^4, \mathbf{y}^5)].$$

When the underlying graphical structure of the output space has a low tree-width, it is still possible to efficiently evaluate the objective of problem (2.19), since the following problem lends itself to efficient optimization:

$$\bar{\mathbf{y}}_i = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}_i} \Delta_i(\mathbf{y}) - \mathbf{w}^\top \Psi_i(\mathbf{y}) \tag{2.21}$$

Note that the value of ξ_i can be computed using $\overline{\mathbf{y}}_i$ as $\xi_i = \Delta_i(\overline{\mathbf{y}}_i) - \mathbf{w}^\top \Psi_i(\overline{\mathbf{y}}_i)$. We refer to the above problem as the *max-oracle*. Let $P(\mathbf{y})$ denote the probability distribution represented by the graph parameterized by the loss augmented scores as potential functions. The max-oracle gives the most probable configuration of this distribution $P(\mathbf{y})$. It has been shown through several works, including cutting-plane algorithms [48], subgradient descent [81] and Frank-Wolfe [43], that an efficient solution to the above problem is sufficient to solve problem (2.19) and/or its Lagrangian dual (2.20) efficiently. As we will see shortly, our work exploits the fact that, for low tree-width graphs, a related problem known as the *expectation-oracle* can be solved efficiently as well (with the same time complexity as the max-oracle). While the max-oracle gives the most probable configuration, the expectation-oracle gives the marginals for the parts over which the output space decomposes. For example, the marginal for part $\mathbf{y}^q$ can be represented as $\mu_i(\mathbf{y}^q) = \sum_{\mathbf{y}^l \neq \mathbf{y}^q} P(\mathbf{y}^1, ..., \mathbf{y}^p)$. Similarly, the marginal for a clique $\mathcal{C} = \{\mathbf{y}^q, \mathbf{y}^r\}$ would be $\mu_i(\mathbf{y}^q, \mathbf{y}^r) = \sum_{\mathbf{y}^l \notin \{\mathbf{y}^q, \mathbf{y}^r\}} P(\mathbf{y}^1, ..., \mathbf{y}^p)$. By cleverly exploiting this observation, we obtain a natural generalization of the Frank-Wolfe algorithm that retains many of its desirable properties such as guaranteed descent direction, analytically computable optimal step size and guaranteed convergence even in block-coordinate mode, while allowing the use of the expectation-oracle to find a valid descent direction that can often lead to improved performance in practice.

The partial-linearization algorithm for optimizing the dual multi-class SVM problem is outlined in Algorithm 1. Step 6 in Algorithm 1, requires us to explicitly compute the update direction corresponding to each dual variable. When the number of dual variables are small as in the case of multiclass classification, this step can be performed efficiently. However, in general, the number of dual variables is proportional to the size of the output space $\mathcal{Y}$. So, when $|\mathcal{Y}|$ is big, it becomes computationally infeasible to explicitly compute the update direction for each dual variable. When the temperature hyperparameter $\tau = 0$, that is when partial-linearization reduces to Frank-Wolfe, this deficiency can be alleviated using an equivalent algorithm that updates the primal variables only. We now show that such an efficient implementation is actually possible for all values of the temperature hyperparameter.

The key observation behind an efficient implementation of partial-linearization using primal variables is that we are only required to compute the marginals of the output variables. Recall from the above discussion that when the output space $\mathcal{Y}_i$ has the underlying graph structure of a tree or a low tree-width graph, it is possible to efficiently compute the exact marginals of the parts of output space, over which it decomposes, by solving the *expectation-oracle* problem. This can be done using a message passing algorithm over a junction tree corresponding to the underlying graph of the output space [100]. For higher tree-width graph structures, although it is not generally possible to efficiently compute the exact marginals, we can get good approximations efficiently.

In order to compute the required marginals with respect to the distribution $\mathbf{s}_i^k$, we parameterize the underlying graph of the output space by node potentials:

$$\theta_i^{s^k}(\mathbf{y}^q) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q) - {\mathbf{w}^{k-1}}^\top \Psi_i(\mathbf{y}^q))$$

and edge potentials:

$$\begin{aligned}\theta_i^{s^k}(\mathbf{y}^q, \mathbf{y}^r) = \quad & \theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) \\ & + \tfrac{1}{\tau}(\Delta_i(\mathbf{y}^q, \mathbf{y}^r) - {\mathbf{w}^{k-1}}^\top \Psi_i(\mathbf{y}^q, \mathbf{y}^r)).\end{aligned}$$

Where, $\theta_i^{\alpha^{k-1}}$ are the potential functions corresponding to the marginals from the $(k-1)^{th}$ iteration. They can be computed from the marginals as $\theta_i^{\alpha^{k-1}}(\mathbf{y}^q) = (1 - \deg(\mathbf{y}_q)) \log(\mu_i^k(\mathbf{y}_q))$ for the nodes and as $\theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) = \log(\mu_i^k(\mathbf{y}_q, \mathbf{y}_r))$ for the edges. Here, $\deg(\mathbf{y}_q)$ denotes the degree of the node $\mathbf{y}^q$.

Proposition 4. *For tree structured output space, given the marginals for the parts, the parameter vector* $\mathbf{w}$ *can be computed as,*

$$\begin{aligned}\mathbf{w} = \quad & \textstyle\sum_i \sum_{\mathbf{y}^q} \mu_i(\mathbf{y}^q)\Psi_i(\mathbf{y}^q) \\ & + \textstyle\sum_i \sum_{\mathbf{y}^q, \mathbf{y}^r} \mu_i(\mathbf{y}^q, \mathbf{y}^r)\Psi_i(\mathbf{y}^q, \mathbf{y}^r) \qquad (2.22)\end{aligned}$$

Proof. As we discuss the case in which the output space has a tree structure, any output can be decomposed on to a set of parts $p \in P$. P being the set of parts. Let r_p be a configuration a part can have from a set of all possible configurations R_p for the part p. Then any output vector $\mathbf{y}$ can be composed of a combination of $r \in R_p$ for the different parts $p \in P$. The feature vector $\Psi_i(\mathbf{y})$ can also be decomposed into a sum of feature vectors for individual parts.

$$\Psi_i(\mathbf{y}) = \sum_{p \in P} \Psi_i(r_p)$$

We define θ_i^k as the potentials computed at iteration k for sample i and σ_y as the mapping from potentials to the distribution.

$$\boldsymbol{\alpha}_{iy} = \sigma_y(\theta_i^k)$$

We can compute the weight vector $\mathbf{w}$ for the corresponding vector $\mathbf{s}^k$ in the dual space as follows.

$$\begin{aligned}
\mathbf{w}(\mathbf{s}^k) &= \sum_i \sum_y s_{iy}^k \left(\Psi_i(\mathbf{y})\right) \\
&= \sum_i \sum_y s_{iy}^k \Psi_i(\mathbf{y}) \\
&= \sum_i \sum_y \sigma_y(\theta_i^k) \Psi_i(\mathbf{y}) \\
&= \sum_i \sum_y \sigma_y(\theta_i^k) \sum_{r_p \in y} \Psi_i(r_p) \\
&= \sum_i \sum_y \sum_{r_p \in y} \sigma_y(\theta_i^k) \Psi_i(r_p) \\
&= \sum_i \sum_{p \in P} \sum_{r_p \in R_p} \left(\sum_{y : r_p \in y} \sigma_y(\theta_i^k) \right) \Psi_i(r_p) \\
&= \sum_i \sum_{p \in P} \sum_{r_p \in R_p} \mu_{i,r_p}(\theta_i^k) \Psi_i(r_p) \qquad (2.23)
\end{aligned}$$

Here, $\mu_{i,r_p}(\theta_i^k)$ is the marginal probability of the configuration r_p of part p for sample i in the k^{th} iteration. $\square$

The complete partial linearization based optimization algorithm involving iterative update of the primal variables is outlined in Algorithm 3. In the k^{th} iteration of the algorithm, we compute the marginals μ_i^k for a chosen sample i (steps 5-7). We first compute the potentials θ_i^k to parameterize the graph underlying the output space. The marginals μ_i^k are then computed by a message passing algorithm over the graph. In step 8, making use of proposition 3, we compute the primal vector $\mathbf{w}_s$ in the update direction from the marginals. We also compute the expected loss l_s using the marginals in step 9. Next, we compute the optimal update step γ (step 10) and update the primal variable vector $\mathbf{w}$, the loss $\mathbf{l}$ and the marginals μ in steps 11-15. The epochs are repeated until the convergence of the dual objective.

2.5.2 Numerically stable implementation of sum-product Belief Propagation

During the training process, we can encounter situations in which the range of the values of the potentials associated with the graph structure is very big. Computing factors by exponentiating these

Algorithm 3 *Block-Coordinate Partial-linearization with primal variable update for optimizing* SSVM

```
1: $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$
2: Initialize marginals $\mu^0$ such that $\mathbf{w}(\mu^0) \sim [0]^d$, $k \leftarrow 1$, $l^0 \leftarrow 0$ and $\forall i \in [n]$, $l_i^0 \leftarrow 0$
3: Initialize a $(d \times n)$ matrix $W$ such that $i^{th}$ column of $W$, $\mathbf{w}_i = \mathbf{w}(\mu_i^0)$
4: **repeat**
5:     Chose a random $i \in [n]$
6:     Compute potential functions to parameterize the graph:
       $\forall$ nodes $\mathbf{y}^q$,
       $\theta_i^{s^k}(\mathbf{y}^q) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q) - {\mathbf{w}^{k-1}}^\top \Psi_i(\mathbf{y}^q))$
       and $\forall$ edges $(\mathbf{y}^q, \mathbf{y}^r)$,
       $\theta_i^{s^k}(\mathbf{y}^q, \mathbf{y}^r) = \theta_i^{\alpha^{k-1}}(\mathbf{y}^q, \mathbf{y}^r) + \frac{1}{\tau}(\Delta_i(\mathbf{y}^q, \mathbf{y}^r) - {\mathbf{w}^{k-1}}^\top \Psi_i(\mathbf{y}^q, \mathbf{y}^r))$.
7:     Compute marginals $\mu_s^k(i)$ by doing message passing over the graph
       parameterized by potentials $\theta_i^{s^k}$.
8:     $\mathbf{w}_s \leftarrow \sum_{\mathbf{y}^q} \mu_i^k(\mathbf{y}^q)\Psi_i(\mathbf{y}^q) + \sum_{\mathbf{y}^q, \mathbf{y}^r} \mu_i^k(\mathbf{y}^q, \mathbf{y}^r)\Psi_i(\mathbf{y}^q, \mathbf{y}^r)$
9:     $l_s \leftarrow \sum_{\mathbf{y}}^q \mu_i^k(\mathbf{y}^q)\Delta_i(\mathbf{y}^q)$
10:    Optimal step size,
       $\gamma \leftarrow \frac{\lambda < \mathbf{w}_i^k, \mathbf{w}_i^k - \mathbf{w}_s > - (l_i^k - l_s)}{\lambda ||\mathbf{w}_i^k - \mathbf{w}_s||^2}$
11:    Update $\mathbf{w}_i$: $\mathbf{w}_i^k \leftarrow (1-\gamma)\mathbf{w}_i^{k-1} + (\gamma)\mathbf{w}_s^k$
12:    Update $l_i$: $l_i^k \leftarrow (1-\gamma)l_i^{k-1} + (\gamma)l_s^k$
13:    Update $\mathbf{w}$: $\mathbf{w}^k \leftarrow \mathbf{w}^{k-1} - \mathbf{w}_i^{k-1} + \mathbf{w}_i^k$
14:    Update $l$: $l^k \leftarrow l^{k-1} - l_i^{k-1} + l_i^k$
15:    Update the marginals: $\mu^k = (\gamma)\mu_s^k + (1-\gamma)\mu^{k-1}$
16:    $k \leftarrow k + 1$
17: **until** Convergence
18: Optimal parameter, $\mathbf{w}$
```

potentials would inadvertently lead to underflow or overflow. Overflow leads to numerical infinities which are difficult to handle. Whereas, underflow leads to factors getting truncated to 0, which leads to dual variables getting stuck on facets of the domain polytope. In order to avoid such kind of numerical instability, we try to do all our computations in the log-space or the potential space, as far as possible.

We shall illustrate our method with a simple example. Consider a tree with 2 nodes $\{u_1, u_2\}$ and a single edge e_{12}. Let x_i denote the random variable associated with node u_i. Then the unary potentials associated with node u_i is denoted by $\theta(x_i)$ and the pairwise potential associated with the edge e_{12} is denoted by $\theta(x_1, x_2)$. In the belief propagation algorithm, let the message that is passed from node u_i to node u_j be denoted by $msg_{ij}(x_j)$. We compute the message $msg_{21}(x_1)$ as follows,

$$msg_{21}(x_1) = \sum_{x_2} \exp\left(\theta_2(x_2) + \theta_{12}(x_1, x_2) - C_1(x_1)\right) \tag{2.24}$$

Where, $C_1(x_1) = \max_{x_2}\left(\theta_2(x_2) + \theta_{12}(x_1, x_2)\right) - M$, M being a number chosen according to the maximum representational capacity of the machine. This clamping operation before exponentiation, guarantees that at least one element inside the summation in equation 2.24 is equal to M. Hence, the message never gets truncated to 0. Once all the messages are computed, the log-marginals for example $\log(P(x_1))$ are computed as follows,

$$\log\left(P(x_1)\right) = \theta_1(x_1) + \log\left(msg_{21}(x_1)\right) + C_1(x_1) - \log(z) \tag{2.25}$$

Where, $\log(z)$ is the log-partition function. This method requires us to store the clamping constants $C_i(x_i)$'s along with the messages. This implementation allows us to run our optimization algorithm for very low values of lambda and temperature without any numerical instability.

2.6 Experiments

We now demonstrate the efficacy of our algorithm on the challenging multi-class classification tasks of action classification, object recognition and gesture recognition. We also present some preliminary results for tree-structured models on the task of handwritten word recognition and scene text recognition.

2.6.1 Results for Multi-class SVM

2.6.1.1 Datasets and Tasks

Action Classification

Dataset. We use the PASCAL VOC 2011 [30] action classification dataset for our experiments. This dataset consists of 4846 bounding boxes of persons, each of which is labeled using one of ten action classes. It includes 3347 'trainval' person bounding boxes for which the ground-truth action classes are known.

Modelling and Features. We train a multi-class SVM as an action classifier using 2800 labelled bounding boxes from the 'trainval' set. We use the standard poselet [62] activation features as sample feature for each person bounding box. The feature vector consists of 2400 action poselet activations and 4 object detection scores. We refer the reader to [62] for details regarding the feature vector.

Object Recognition on CIFAR-10 dataset

Dataset. We use the CIFAR-10 dataset [57] for this set of experiments. It consists of a total of 60,000 images of 10 different object classes with 6,000 images per class. The dataset is divided into a 'trainval' set of 50,000 images and a 'test' set of 10,000 images.

Modelling and Features. We train a multi-class SVM for object recognition on the trainval set. To represent each image, we use a feature representation that is extracted from a trained Convolutional Neural Network. Specifically, we pass the resized image as input to the VGG-NET [85] network and use the activation vector of its penultimate layer as the feature vector. The length of the resulting feature vector is 4096.

Object Recognition on PASCAL VOC dataset

Dataset. We use the PASCAL VOC 2007 [29] object detection dataset, which consists of a total of 9963 images of which 5011 images are in the 'trainval' set. All the images are labelled to indicate the presence or absence of the instances of 20 different object categories. Each image can have multiple instances of an object and we are provided with tight bounding boxes around each of them.

Modelling and Features. We train a multi-class SVM for object recognition on 12,608 object bounding boxes extracted from the trainval set. For each object bounding box, we use a feature representation extracted from a trained Convolutional Neural Network (CNN). Specifically, we pass the bounding box as input to the CNN and use the activation vector of the penultimate layer of the CNN as the feature vector. Inspired by the work of Girshick *et al.* [35], we use the CNN that is trained on the ImageNet dataset [21], by rescaling each window to a fixed size of 224×224. The length of the resulting feature vector is 4096.

Gesture Recognition

Dataset. We use the MSRC-12 data set [33] which contains 594 sequences of motion capture data obtained using a Kinect sensor. Each sequence corresponds to a person repeatedly performing one out of the 12 gestures represented in the dataset. For each frame of the sequence, we are given the 3D world position of 20 human body joints. In addition to the sequence level gesture annotations, we are also provided with frame level annotations which we ignore in our experiments.

Modelling and Features. We treat each sequence as a single sample and train a multi-class latent-SVM for sequence level gesture recognition. The exact location of the gesture in a sequence is held by a latent variable. We represent a sequence **x** using a feature vector $\phi(\mathbf{x}, h)$ which is extracted from the frame in the sequence denoted by the latent variable h. We extract the same 130 dimensional feature vector from a frame as used in [33].

2.6.1.2 Methods

For all the tasks, we compare the runtime of our block-coordinate partial linearization (BCPL) approach to those of two baseline algorithms for solving the multi-class SVM, namely the block-coordinate Frank-Wolfe algorithm [43] (BCFW) and the online exponentiated gradient (OEG) algorithm [16]. We ran each of the algorithms for 3 different values $(0.1, 0.01, 0.001)$ of the regularization parameter λ. For most practical setups λ is chosen to be very low since large datasets avoid the problem of high generalization error via overfitting. In all the experiments, we used a fixed temperature of $\tau = 0.01$ for our algorithm. For OEG, we repeated the experiments for 8 different values (100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 10^{-5}, 10^{-10}) of the temperature parameter τ and report the results for the best performing value. We initialize all the optimization algorithms in a manner which ensures that the weight parameters are almost equal to 0. In each iteration of training, we sample without repetition from the dataset. For the BCPL and OEG algorithms, in order to avoid getting stuck on a facet of the domain polytope, we truncate the step size γ at each iteration to $1 - \epsilon$. Where, $\epsilon = 2.2204 \times 10^{-16}$ is the machine epsilon.

2.6.1.3 Results

We report the performance of the different methods in terms of the increase in the dual MC-SVM objective function with respect to training time. Figure 2.3 provides the detailed plots for the experiments for different values of λ. As can be observed from the plots, in most cases, our BCPL algorithm converges faster than BCFW and OEG. It should be noted that the relative difference between the rate of convergence of the two algorithms may seem comparatively small. However, due to the low absolute rate of convergence of both the algorithms in the later stages, this small gap leads to significant saving

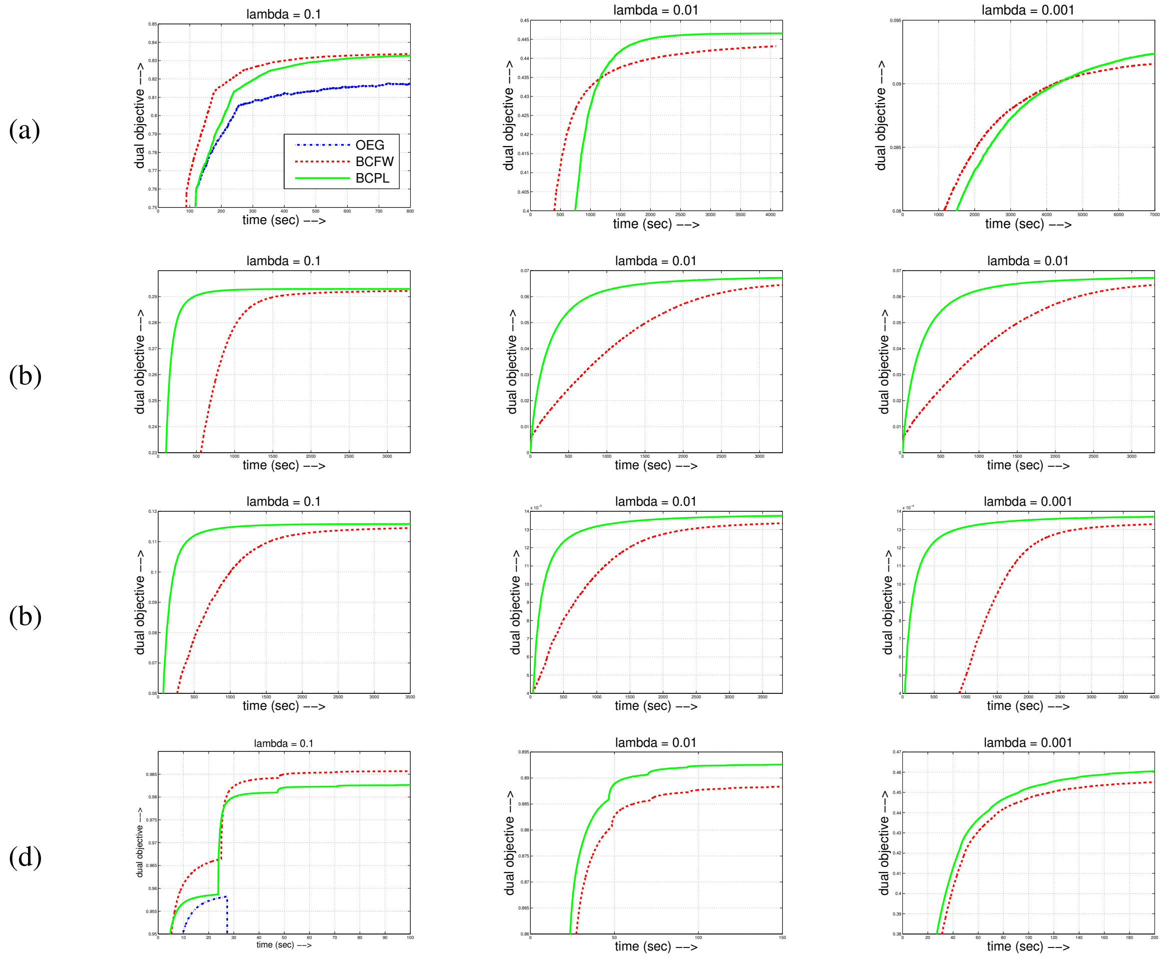
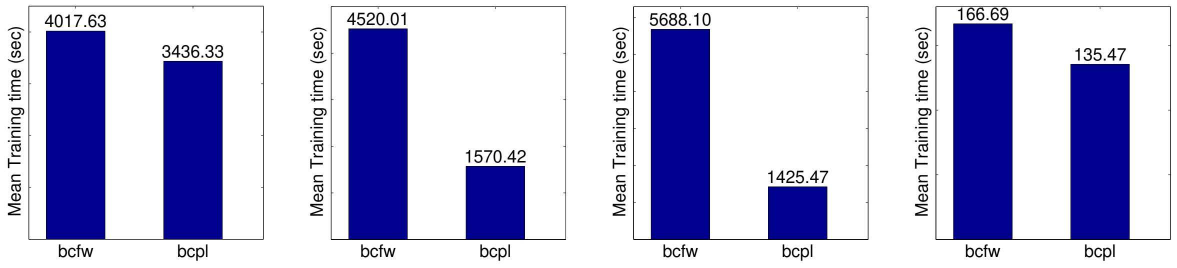


Figure 2.3: *Comparison of different optimization algorithms for the multi-class* SVM *learning problem in terms of change in the dual objective (negative of the objective of problem (2.4)) with respect to training time. The results correspond to (a) Action classification (b) Object recognition on* CIFAR-*10 (c) Object recognition on* PASCAL VOC *(d) Gesture recognition. The figures are zoomed-in along the vertical axis to highlight the differences between the top most competing methods. Note that for* $\lambda = 0.01$ *and* $\lambda = 0.001$*, the exponentiated gradient algorithm performs significantly worse than the other two methods, and is therefore not visible in the plots. This figure is best viewed in colour.*



(a) (b) (c) (d)

Figure 2.4: *Comparison of Block-coordinate Frank-Wolfe (*BCFW*) and Block-coordinate Partial linearization (*BCPL*) in terms of the mean training time. The results correspond to (a) Action classification (b) Object recognition on* CIFAR-*10 (c) Object recognition on* PASCAL VOC *(d) Gesture recognition.*

in terms of iterations and time for our algorithm. The OEG algorithm performs consistently worse than the other 2 algorithms for these set of experiments. For all the tasks, we also report the mean time taken for training by our method and the Frank-Wolfe algorithm. For each task, the training time is averaged over all values of λ. Figure 2.4 shows that our approach consistently does better than the Frank-Wolfe algorithm. Note that since we solve a convex optimization problem, all the methods are guaranteed to converge to the same or very similar solutions. Hence, we have focused on only a comparison of the run time here.

2.6.2 Results for Structured SVM

2.6.2.1 Datasets and Tasks

Handwritten Text Recognition

Dataset. We use the OCR dataset [91] for our experiments. The dataset consists of 6251 images of handwritten words. We use 626 images for training and the rest for testing. Each word image is already segmented into individual characters. Each character can be of one of the 26 classes: $\{a, ..., z\}$.

Modelling and Features. The dataset provides the handwritten-word images in binary format. Each segmented character image in the dataset is of size 16×8 pixels. We use binary pixel values of the character images to construct a 128 dimensional feature vector for each character. We use an indicator basis function to represent the correlation between adjacent characters. We also use indicator basis functions to represent location independent bias for each of the characters and additional bias for the first and the last characters of any word. This makes the overall size of the feature vector equal to $(128 \times 26 + 26 \times 26 + 26 + 26 \times 2) = 4082$. Note that the underlying graph has a 'chain' structure, which enables the computation of exact marginals via sum-product belief propagation [100].

Scene-text recognition

Dataset. We use the IIIT-5k dataset [65] for the scene-text recognition problem. The dataset consists of 5000 word images collected from natural scenes. It includes 2000 'trainval' and 3000 'test' images. We are also given the bounding boxes of the characters in each word image. Each character can be of one of the 62 classes: $\{a, ..., z, A, ..., Z, 0, ..., 9\}$.

Modelling and Features. The word images in the dataset are of different sizes. Consequently, the size of the character bounding boxes vary over a wide range. We run 2 sets of experiments by resizing each of the segmented character images first to the size of 50×25 pixels and then to 70×35 pixels. We represent each character image using either a 1250 sized or a 2450 sized feature vector constructed from its gray-scale pixel values. Similar to the handwritten word recognition problem, we use indicator basis functions to represent the correlation between adjacent characters, location independent bias for each of the characters and additional bias for the first and the last characters of any word. This makes the overall size of the feature vector equal to $(1250 \times 26 + 26 \times 26 + 26 + 26 \times 2) = 33254$ for character image size of 50×25 and $(2450 \times 26 + 26 \times 26 + 26 + 26 \times 2) = 64454$ for character image size of 70×35.

2.6.2.2 Methods

We compare our block-coordinate partial linearization (BCPL) algorithm with the block-coordinate Frank-Wolfe algorithm [43] (BCFW) and the online exponentiated gradient (OEG) algorithm [16]. For the scene text recognition task, we repeated the same set of experiments for both character image size of 50×25 as well as 70×35. We ran each method for 2 values $(0.1, 0.01)$ of λ. For the handwritten text recognition task, we used a fixed temperature of $\tau = 0.01$ for our algorithm, same as in case of the multi-class SVM experiments. Whereas for the scene text recognition task, we repeat the BCPL algorithm for 5 values $(1, 0.1, 0.01, 0.001, 10^{-5})$ of τ and report the best results. Similar to the multi-class SVM experiments, we initialize all the optimization algorithms in a manner which ensures that the weight parameters are almost equal to 0 and for the BCPL and OEG algorithms, also truncate the step size γ at each iteration to $1 - \epsilon$.

2.6.2.3 Results

We report the performance of the different optimization algorithms for the structured SVM learning problem. Figure 2.5 shows the progress of the optimization algorithms over time. For the handwritten text recognition problem, our BCPL algorithm converges faster than BCFW and OEG. For the scene-text recognition problem, while for a character image size of 50×25, BCFW has a faster rate of convergence than our method, for 70×35, our method performs better and beats BCFW for $\lambda = 0.1$. In all the experiments, while the number of iterations required is always lower for our method, the time taken by each iteration can be significantly more for each iteration for some problems. In general for a given problem, our method performs better when the size of the feature vector is large. It's quite common to have large feature vectors for many practical problems particularly those related to computer vision and are potential candidates for application of our optimization algorithm.

2.6.3 Comparison between Partial-linearization at low temperatures and Frank-Wolfe

For very low values ($\leq 10^{-5}$) of temperature, as can be seen in figure 2.6, BCPL behaves almost exactly same as BCFW when change in objective function value is considered over number of iterations. The small gap observed between the BCFW and BCPL at low temperature is because of the difference in running time of the oracles for the respective algorithms.

2.7 Discussion

We proposed a partial linearization based approach for optimizing multi-class SVM, which naturally generalizes the Frank-Wolfe and the exponentiated gradient algorithms. Our method introduces the key temperature hyperparameter for which we keep a fixed value through out the optimization. This leaves scope for exploring ideas for varying the temperature across iterations for faster convergence. In this work, we discussed our approach only in context of multi-class classification models and structured SVM models that have a tree structure. However, the efficacy of our approach in the context of loopy graphs that require approximate evaluation of the expectation oracle is still unknown. Another interesting direction for future research would be to explore the applicability of our approach for variations of the

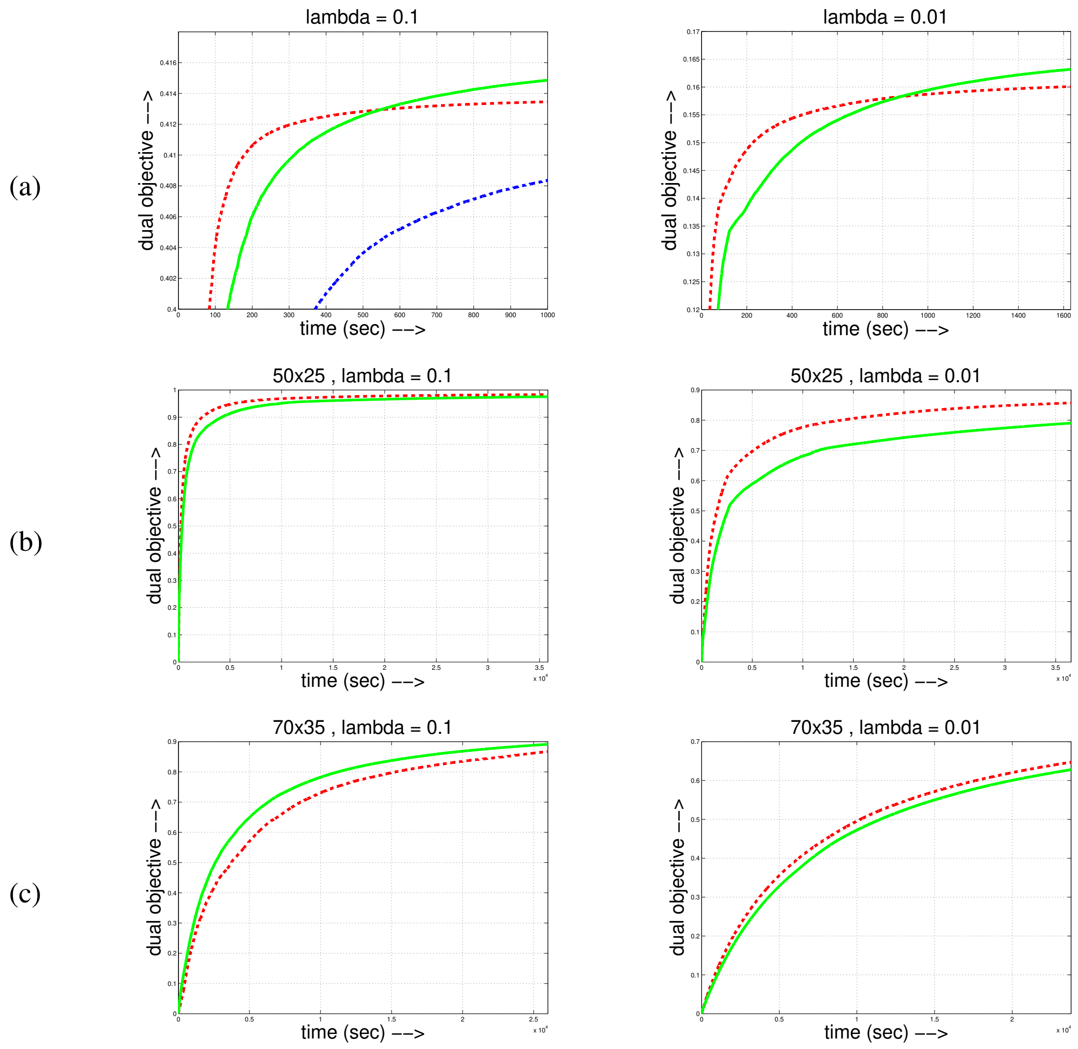
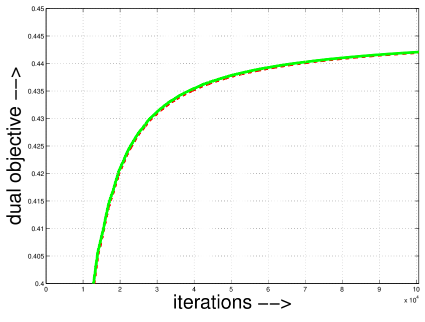
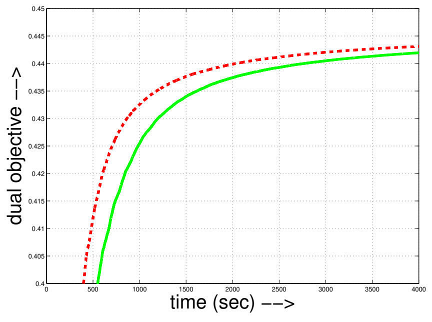


Figure 2.5: *Comparison of optimization algorithms for the structured* SVM *learning problem in terms of change in the dual objective with respect to training time. The results correspond to (a) Handwritten text recognition (b) Scene-text recognition with character image size of* 50×25 *(c) Scene-text recognition with character image size of* 70×35*. Here, the broken red curves correspond to the Frank-Wolfe algorithm, solid green to our partial-linearization algorithm. Note in most of the cases, the exponentiated gradient algorithm performs significantly worse than the other two methods, and is therefore not visible in the plots. This figure is best viewed in colour.*



(a)



(b)

Figure 2.6: *Comparison of the Frank-Wolfe algorithm with Partial-linearization at very low temperature ($\tau = 10^{-10}$). Left plot shows change in dual objective over training iterations where as the right plot shows the change in dual objective over time. Here, the broken red curves correspond to the Frank-Wolfe algorithm and the solid green to our partial-linearization algorithm with* $\tau = 10^{-10}$.

SVM optimization problem (such as those that use soft constraints), or for other learning frameworks such as neural networks.

Chapter 3

Efficient optimization of rank-based loss functions

3.1 Introduction

Information retrieval systems require us to rank a set of samples according to their relevance to a query. The risk of the predicted ranking is measured by a user-specified loss function. Several intuitive loss functions have been proposed in the literature. These include simple decomposable losses (that is, loss functions that decompose over each training sample) such as 0-1 loss [60, 70] and the area under the ROC curve [3, 39], as well as the more complex non-decomposable losses (that is, loss functions that depend on the entire training data set) such as the average precision (AP) [13, 105] and the normalized discounted cumulative gain (NDCG) [14].

When learning a retrieval system, one can use a training objective that is agnostic to the risk, such as in the case of LambdaMART [12]. In this work, we focus on approaches that explicitly take into account the loss function used to measure the risk. Such approaches can use any one of the many machine learning models such as structured support vector machines (SSVM) [90, 93], deep neural networks [88], decision forests [51], or boosting [82]. To estimate the parameters of the model, it is common to employ a training objective that is related to the empirical risk associated with a rank-based loss function. Many of the rank-based loss functions that we are interested in happen to be non-decomposable, that is, they can not be expressed as summation of terms dependent on individual samples.

There have been considerable amount of prior work towards designing of efficient methods for optimizing non-decomposable rank-based loss functions. A wide array of work has been focused on encoding the loss function in the form of constraints on classification rates for which then differentiable surrogates are employed to allow for gradient based optimization [18, 25, 37, 72, 74]. Our approach is more closely related to methods that involve designing convex surrogates for a specific class of rank-based loss functions [49, 73, 75, 105]. Specifically, we focus on a structured hinge upper bound to the rank-based loss function [14, 105], or an asymptotic alternative such as direct loss minimization [38, 86].

The feasibility of both the structured hinge loss and the direct loss minimization approach depends on the computational efficiency of the *loss-augmented inference* procedure. When the loss function is decomposable, the loss-augmented inference problem can be solved efficiently by independently considering each training sample. However, for non-decomposable loss functions, it presents a hard computational challenge. For example, given a training data set with P positive (relevant to the query) and N negative (not relevant to the query) samples, the best known algorithms for loss-augmented inference for AP and NDCG loss functions have a complexity of $O(PN + N \log N)$ [14, 105]. Since the number of negative samples N can be very large in practice, this prohibits their use on large data sets.

Contributions. In order to address the computational challenge of non-decomposable loss functions such as those based on AP and NDCG, we make three contributions.

- We characterize a large class of ranking based loss functions that are amenable to a novel quicksort flavored optimization algorithm for the corresponding loss-augmented inference problem. We refer to this class of loss functions as QS-*suitable*.
- We show that the AP and the NDCG loss functions are QS-suitable, which allows us to reduce the complexity of the corresponding loss-augmented inference to $O(P \log N + N \log P)$.
- We prove that there cannot exist a comparison based method for loss-augmented inference that can provide a better asymptotic complexity than our quicksort flavored approach.

For the sake of clarity, we limit our discussion to the structured hinge loss upper bound of the loss function. However, as our main contribution is to speed-up loss-augmented inference, it is equally applicable to direct loss minimization. We demonstrate the efficacy of our approach on the challenging problems of action recognition, object detection and image classification, using publicly available data sets. Rather surprisingly, we show that in case of some models, parameter learning by optimizing complex non-decomposable AP and NDCG loss functions can be carried out faster than by optimizing simple decomposable 0-1 loss. Specifically, while each loss-augmented inference call is more expensive for AP and NDCG loss functions, it can take fewer calls in practice to estimate the parameters of the corresponding model.

3.2 Background

We begin by providing a brief description of a general retrieval framework that employs a rank-based loss function, hereby referred to as the ranking framework. Note that this framework is the same as or generalizes the ones employed in previous works [14, 45, 86, 105]. The two specific instantiations of the ranking framework that are of interest to us employ the average precision (AP) loss and the normalized discounted cumulative gain (NDCG) loss respectively. A detailed description of the two aforementioned loss functions is provided in the subsequent subsection.

3.2.1 The Ranking Framework

Input. The input to this framework is a set of n samples, which we denote by $\mathbf{X} = \{\mathbf{x}_i; i = 1, \ldots, n\}$. For example, each sample can represent an image and a bounding box of a person present in the image. In addition, we are also provided with a query, which in our example could represent an action such as 'jumping'. Each sample can either belong to the positive class (that is, the sample is relevant to the query) or the negative class (that is, the sample is not relevant to the query). For example, if the query represents the action 'jumping' then a sample is positive if the corresponding person is performing the jumping action and negative otherwise. The set of positive and the negative samples are denoted by $\mathcal{P}$ and $\mathcal{N}$ respectively, which we assume are provided during training but are not known during testing.

Output. Given a query and a set of n samples $\mathbf{X}$, the desired output of the framework is a ranking of the samples according to their relevance to the query. This is often represented by a ranking matrix $\mathbf{R} \in \{-1, 0, 1\}^{n \times n}$ such that $\mathbf{R}_{\mathbf{x},\mathbf{y}} = 1$ if $\mathbf{x}$ is ranked higher than $\mathbf{y}$, -1 if $\mathbf{x}$ is ranked lower than $\mathbf{y}$ and 0 if $\mathbf{x}$ and $\mathbf{y}$ are ranked the same. In other words, $\mathbf{R}$ is an anti-symmetric matrix that represents the relative ranking of a pair of samples.

Given the sets $\mathcal{P}$ and $\mathcal{N}$ during training, we construct a ground truth ranking matrix $\mathbf{R}^*$, which ranks each positive sample above all the negative samples. Formally, the ground truth ranking matrix $\mathbf{R}^*$ is defined such that $\mathbf{R}^*_{\mathbf{x},\mathbf{y}} = 1$ if $\mathbf{x} \in \mathcal{P}$ and $\mathbf{y} \in \mathcal{N}$, -1 if $\mathbf{x} \in \mathcal{N}$ and $\mathbf{y} \in \mathcal{P}$, and 0 if $\mathbf{x}, \mathbf{y} \in \mathcal{P}$ or $\mathbf{x}, \mathbf{y} \in \mathcal{N}$. Note that the ground truth ranking matrix only defines a partial ordering on the samples since $\mathbf{R}^*_{i,j} = 0$ for all pairs of positive and negative samples. We will refer to rankings where no two samples are ranked equally as *proper rankings*. Without loss of generality, we will treat all rankings other than the ground truth one as a proper ranking by breaking ties arbitrarily.

Discriminant Function. Given an input set of samples $\mathbf{X}$, the discriminant function $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ provides a score for any candidate ranking $\mathbf{R}$. Here, the term **w** refers to the parameters of the discriminant function. We assume that the discriminant function is piecewise differentiable with respect to its parameters **w**. One popular example of the discriminant function used throughout the ranking literature is the following:

$$F(\mathbf{X}, \mathbf{R}; \mathbf{w}) = \frac{1}{|\mathcal{P}|\,|N|} \sum_{\mathbf{x} \in \mathcal{P}} \sum_{\mathbf{y} \in \mathcal{N}} \mathbf{R}_{\mathbf{x},\mathbf{y}} (\phi(\mathbf{x}; \mathbf{w}) - \phi(\mathbf{y}; \mathbf{w})). \tag{3.1}$$

Here, $\phi(\mathbf{x}; \mathbf{w})$ is the score of an individual sample, which can be provided by a structured SVM or a deep neural network with parameters **w**.

Prediction. Given a discriminant function $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ with parameters **w**, the ranking of an input set of samples $\mathbf{X}$ is predicted by maximizing the score, that is, by solving the following optimization problem:

$$\mathbf{R}(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{R}} F(\mathbf{X}, \mathbf{R}; \mathbf{w}). \tag{3.2}$$

The special form of the discriminant function in equation (3.1) enables us to efficiently obtain the predicted ranking $\mathbf{R}(\mathbf{w})$ by sorting the samples in descending order of their individual scores $\phi(\mathbf{x}; \mathbf{w})$. We refer the reader to [45, 105] for details.

Parameter Estimation. We now turn towards estimating the parameters of our model given input samples $\mathbf{X}$, together with their classification into positive and negative sets $\mathcal{P}$ and $\mathcal{N}$ respectively. To this end, we minimize the risk of prediction computed using a user-specified loss function $\Delta(\mathbf{R}^*, \mathbf{R}(\mathbf{w}))$, where $\mathbf{R}^*$ is the ground truth ranking that is determined by $\mathcal{P}$ and $\mathcal{N}$ and $\mathbf{R}(\mathbf{w})$ is the predicted ranking as shown in equation (3.2). We estimate the parameters of our model as

$$\mathbf{w}^* = \min_{\mathbf{w}} \mathbb{E}[\Delta(\mathbf{R}^*, \mathbf{R}(\mathbf{w}))]. \tag{3.3}$$

In the above equation, the expectation is taken with respect to the data distribution.

Optimization for Parameter Estimation. For many intuitive rank based loss functions such as AP loss and NDCG loss, owing to their non-differentiability and non-decomposability, problem (3.3) can be difficult to solve using simple gradient based methods. One popular approach is to modify problem (3.3) to instead minimize a structured hinge loss upper bound to the user-specified loss. We refer the reader to [105] for further details about this approach.

Formally, the model parameters can now be obtained by solving the following problem:

$$\mathbf{w}^* = \min_{\mathbf{w}} \mathbb{E}[J(\mathbf{w})] \tag{3.4}$$
$$J(\mathbf{w}) = \max_{\mathbf{R}} \Delta(\mathbf{R}^*, \mathbf{R}) + F(\mathbf{X}, \mathbf{R}; \mathbf{w}) - F(\mathbf{X}, \mathbf{R}^*; \mathbf{w})$$

The function $J(\mathbf{w})$ in problem (3.4) is continuous and piecewise differentiable, and is amenable to gradient based optimization. The semi-gradient [1] of $J(\mathbf{w})$ takes the following form:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} F(\mathbf{X}, \bar{\mathbf{R}}; \mathbf{w}) - \nabla_{\mathbf{w}} F(\mathbf{X}, \mathbf{R}^*; \mathbf{w}), \tag{3.5}$$

$$\text{with,} \quad \bar{\mathbf{R}} = \operatorname*{argmax}_{\mathbf{R}} \Delta(\mathbf{R}^*, \mathbf{R}) + F(\mathbf{X}, \mathbf{R}; \mathbf{w}). \tag{3.6}$$

Borrowing terminology from the structured prediction literature [47, 105], we call $\bar{\mathbf{R}}$ the *most violating ranking* and problem (3.6) as the *loss-augmented inference* problem. An efficient procedure for loss-augmented inference is key to solving problem (3.4).

While we focus on using loss-augmented inference for estimating the semi-gradient, it can also be used as the cutting plane [47] and the conditional gradient of the dual of problem (3.4). In addition to this, loss-augmented inference is also required for solving problem (3.3) using the direct loss minimization framework [86].

3.2.2 Loss Functions

A particular property of the loss functions that affects the ease of optimizing them is the decomposability of the loss function onto samples. Based on this property, loss functions can be categorized into *decomposable* and *non-decomposable* loss functions.

Notation. In order to discuss the optimizability of loss functions, it would be helpful to introduce some additional notation. We define $ind(\mathbf{x})$ to be the index of a sample $\mathbf{x}$ according to the ranking $\mathbf{R}$. Note that the notation does not explicitly depend on $\mathbf{R}$ as the ranking will always be clear from context. If $\mathbf{x} \in \mathcal{P}$ (that is, for a positive sample), we define $ind^+(\mathbf{x})$ as the index of $\mathbf{x}$ in the total order of positive samples induced by $\mathbf{R}$. For example, if $\mathbf{x}$ is the highest ranked positive sample then $ind^+(\mathbf{x}) = 1$ even though $ind(\mathbf{x})$ need not necessarily be 1 (in the case where some negative samples are ranked higher than $\mathbf{x}$). For a negative sample $\mathbf{x} \in \mathcal{N}$, we define $ind^-(\mathbf{x})$ analogously: $ind^-(\mathbf{x})$ is the index of $\mathbf{x}$ in the total order of negative samples induced by $\mathbf{R}$.

[1]For a continuous function $f(x)$ defined on a domain of any generic dimension, we can define semi-gradient $\nabla_s f(x)$ to be a random picking from the set $\{\nabla f(t) : ||x - t|| < \epsilon\}$, for a sufficiently small ϵ.

Decomposable loss function. A loss function is called *decomposable* if it can be written as summation of terms that depend on individual samples. That is, a decomposable loss function $\Delta_d(\mathbf{R}^*, \mathbf{R})$ can be written as:

$$\Delta_d(\mathbf{R}^*, \mathbf{R}) = \sum_{i=1}^{|X|} g_i\left(ind(\mathbf{x}_i)\right).$$

Such loss functions are generally easily optimizable because they can be optimized by independently optimizing each of the terms in the summation that depend only on a single sample:

$$\operatorname*{argmax}_{\mathbf{R}} \Delta_d(\mathbf{R}^*, \mathbf{R}) = \operatorname*{argmax}_{\mathbf{R}} \sum_{i=1}^{|X|} g_i\left(ind(\mathbf{x}_i)\right) = \left[\operatorname*{argmax}_{ind(\mathbf{x}_i)} g_i\left(ind(\mathbf{x}_i)\right); i = 1 \ldots |X|\right]. \quad (3.7)$$

Example of a decomposable loss function is the 0-1 loss. The loss function based on the area under the ROC curve (AUC), though not decomposable onto individual samples, can be decomposed onto pairs of samples.

0-1 Loss. The 0-1 loss is simply equal to the ratio of incorrect retrievals to that of the total number of retrievals. It does not as such depend on the ranking of the retrievals and instead depends on the classification of the retrievals based on a threshold. The 0-1 loss can be additively decomposed onto individual samples as follows:

$$\Delta_{01}(\mathbf{R}^*, \mathbf{R}) = \sum_{i=1}^{|X|} \frac{1}{|X|}\mathcal{I}\left(\mathcal{I}(ind(x_i) \leq ind_{Th}) \neq \mathcal{I}(ind^*(x_i) \leq |\mathcal{P}|)\right).$$

Here, $\mathcal{I}$ is the indicator function and ind_{Th} is the highest index among samples that are classified as correct retrievals.

AUC Loss. AUC is an evaluation measure that is computed as the total area under the receiver operating characteristic (ROC) curve. The AUC loss is simply 1-AUC. Unlike the 0-1 loss, AUC loss is not decomposable onto individual samples. However, it can be additively decomposed onto pairs of samples as follows:

$$\Delta_{AUC}(\mathbf{R}^*, \mathbf{R}) = \sum_{i=1}^{|X|}\sum_{j=1}^{|X|} \frac{\mathcal{I}\left((ind(x_i) \leq ind(x_j)) \; and \; (ind^*(x_i) > ind^*(x_j))\right)}{|\mathcal{P}|\,|\mathcal{N}|}.$$

Non-decomposable loss function. A loss function is called *non-decomposable* if it cannot be written as summation of terms that depend only on individual samples. Many sophisticated loss functions that are strongly sensitive to the ranking order are non-decomposable. Such loss functions are generally difficult to optimize as the problem cannot be broken down into independent simpler optimization problems over individual samples. This includes popular rank-based loss functions like those based on AP and NDCG.

AP Loss. Using the above notation, we can now concisely define the average precision (AP) loss of a proper ranking $\mathbf{R}$ given the ground truth ranking $\mathbf{R}^*$ as follows:

$$\Delta_{AP}(\mathbf{R}^*, \mathbf{R}) = 1 - \frac{1}{|\mathcal{P}|} \sum_{\mathbf{x} \in \mathcal{P}} \frac{ind^+(\mathbf{x})}{ind(\mathbf{x})}.$$

For example, consider an input $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_8\}$ where $\mathbf{x}_i \in \mathcal{P}$ for $1 \leq i \leq 4$, and $\mathbf{x}_i \in \mathcal{N}$ for $5 \leq i \leq 8$, that is, the first 4 samples are positive while the last 4 samples are negative. If the proper ranking $\mathbf{R}$ induces the order

$$(\mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_8, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_2, \mathbf{x}_6, \mathbf{x}_7), \tag{3.8}$$

$$\text{then, } \Delta_{AP}(\mathbf{R}^*, \mathbf{R}) = 1 - \frac{1}{4}\left(\frac{1}{1} + \frac{2}{2} + \frac{3}{4} + \frac{4}{6}\right) \approx 0.146.$$

NDCG Loss. We define a discount $D(i) = 1/\log_2(1+i)$ for all $i = 1, \cdots, |\mathcal{N}| + |\mathcal{P}|$. This allows us to obtain a loss function based on the normalized discounted cumulative gain as

$$\Delta_{NDCG}(\mathbf{R}^*, \mathbf{R}) = 1 - \frac{\sum_{\mathbf{x} \in \mathcal{P}} D(ind(\mathbf{x}))}{\sum_{i=1}^{|\mathcal{P}|} D(i)}.$$

For example, consider the aforementioned input where the first four samples are positive and the last four samples are negative. For the ranking $\mathbf{R}$ that induces the order (3.8), we can compute

$$\begin{aligned} \Delta_{NDCG}(\hat{\mathbf{R}}, \mathbf{R}) = 1 - \frac{1 + \log_2^{-1} 3 + \log_2^{-1} 5 + \log_2^{-1} 7}{1 + \log_2^{-1} 3 + \log_2^{-1} 4 + \log_2^{-1} 5} \\ \approx 0.056. \end{aligned}$$

Both AP loss and NDCG loss are functions of the entire dataset and are not decomposable onto individual samples.

While solving problem (3.6) is non-trivial, especially for non-decomposable loss functions, the method we propose in this work allows for an efficient loss-augmented inference procedure for such complex loss functions. For our discussion, we focus on the average precision (AP) loss and the normalized discounted cumulative gain (NDCG) loss. While the AP loss is very popular in the computer vision community as evidenced by its use in the various challenges of PASCAL VOC [30], the NDCG loss is very popular in the information retrieval community [14].

3.3 Quicksort Flavored Optimization

In order to estimate the parameters $\mathbf{w}$ in the ranking framework by solving problem (3.4), we need to compute the semi-gradient of $J(\mathbf{w})$. To this end, given the current estimate of parameters $\mathbf{w}$, as well as a set of samples $\mathbf{X}$, we are interested in obtaining the most violated ranking by solving problem (3.6). At first glance, the problem seems to require us to obtain a ranking matrix $\bar{\mathbf{R}}$. However, it turns out that we do not explicitly require a ranking matrix.

In more detail, our algorithm uses an intermediate representation of the ranking using the notion of interleaving ranks. Given a ranking $\mathbf{R}$ and a negative sample $\mathbf{x}$, the interleaving rank $rank(\mathbf{x})$ is defined as one plus the number of positive samples preceding $\mathbf{x}$ in $\mathbf{R}$. Note that, similar to our notation for $ind(\cdot)$, $ind^{+}(\cdot)$ and $ind^{-}(\cdot)$, we have dropped the dependency of $rank(\cdot)$ on $\mathbf{R}$ as the ranking matrix would be clear from context. The interleaving rank of all the samples does not specify the total ordering of all the samples according to $\mathbf{R}$ as it ignores the relative ranking of the positive samples among themselves, and the relative ranking of the negative samples among themselves. However, as will be seen shortly, for a large class of ranking based loss functions, interleaving ranks corresponding to the most violating ranking are sufficient to compute the semi-gradient as in equation (3.5).

In the rest of the section, we discuss the class of loss functions that are amenable to a quicksort flavored algorithm, which we call QS-suitable loss functions. We then describe and analyze our quicksort flavored approach for finding the interleaving rank in some detail.

3.3.1 QS-Suitable Loss Functions

As discussed earlier, many popular rank-based loss functions happen to be non-decomposable. That is, they can not be additively decomposed onto individual samples. However, it turns out that a wide class of such non-decomposable loss functions can be instead additively decomposed onto the negative samples. We will call this the *negative-decomposability* property. Further, many of those rank-based loss functions do not depend on the relative order of positive or negative samples among themselves. Rather, the loss for a ranking $\mathbf{R}$, $\Delta(\mathbf{R}^{*}, \mathbf{R})$, depends only on the interleaving rank of positive and negative samples corresponding to $\mathbf{R}$. We will call this the *interleaving-dependence* property. We characterize a class of loss functions, which we call QS-*suitable*, that has such desirable properties. Formally, a proper loss function $\Delta = \Delta(\mathbf{R}^{*}, \mathbf{R})$ is called QS-*suitable* if it meets the following three conditions.

(C1) **Negative decomposability with interleaving dependence.** There are functions $\delta_j\colon \{1,\dots,|\mathcal{P}|+1\} \to \mathbb{R}$ for $j = 1,\dots,|\mathcal{N}|$ such that for a proper ranking $\mathbf{R}$ one can write

$$\Delta(\mathbf{R}^*, \mathbf{R}) = \sum_{\mathbf{x}\in\mathcal{N}} \delta_{ind^-(\mathbf{x})}(rank(\mathbf{x})).$$

(C2) j**-monotonicity of discrete derivative.** For every $1 \le j < |\mathcal{N}|$ and $1 \le i \le |\mathcal{P}|$ we have

$$\delta_{j+1}(i+1) - \delta_{j+1}(i) \ge \delta_j(i+1) - \delta_j(i).$$

(C3) **Fast evaluation of discrete derivative.** For any $j \in \{1,\dots,|\mathcal{N}|\}$ and $i \in \{1,\dots,|\mathcal{P}|\}$, can the value $\delta_j(i+1) - \delta_j(i)$ be computed in constant time.

From (C1), we can see that the loss function depends only on the interleaving ranks of the negative samples. More accurately, it depends on the vector $\mathbf{r} = (r_1,\dots,r_{|\mathcal{N}|})$ where r_i is the interleaving rank of the i-th most relevant negative sample (i.e. with the i-th highest score).

Another way to interpret this type of dependence is by looking at the $\pm$-pattern of a ranking which can be obtained as follows. Given a proper ranking $\mathbf{R}$ (in the form of a permutation of samples), it is the pattern obtained by replacing each positive sample with a "$+$" symbol and each negative sample with a "$-$" symbol. It is easy to see that the $\pm$-pattern uniquely determines the vector $\mathbf{r}$ and vice versa and thus (C1) also implies dependence on the $\pm$-pattern.

As will be evident later in the section, the above properties in a loss function allows for an efficient quicksort flavored divide and conquer algorithm to solve the loss augmented problem. We formally define the class of loss functions that allow for such a quicksort flavored algorithm as QS-*suitable* loss functions. The following propositions establish the usefulness for such a characterization.

Proposition 5. *Δ_{AP} is QS-suitable.*

Proof. Regarding (C1), the functions δ_j were already identified in [105] as

$$\delta_j(i) = \frac{1}{|\mathcal{P}|}\sum_{k=i}^{|\mathcal{P}|}\left(\frac{j}{j+k} - \frac{j-1}{j+k-1}\right) \tag{3.9}$$

so after writing

$$\delta_j(i+1) - \delta_j(i) = \frac{j-1}{j+i-1} - \frac{j}{j+i}$$

we again have (C3) for free and (C2) reduces to

$$2g_i(j) \ge g_i(j-1) + g_i(j+1),$$

where $g_i(x) = \frac{x}{x+i}$, and the conclusion follows from concavity of $g_i(x)$ for $x > 0$. □

Proposition 6. Δ_{NDCG} *is QS-suitable.*

Proof. As for (C1), let us first verify that the functions δ_j can be set as

$$\delta_j(i) = \frac{1}{C}\left(D(i+j-1) - D(|\mathcal{P}|+j)\right),$$

where $C = \sum_{i=1}^{|\mathcal{P}|} D(i)$. Indeed, one can check that

$$\begin{aligned}
&\Delta(\mathbf{R}^*, \mathbf{R}) \\
&= 1 - \frac{\sum_{\mathbf{x}\in\mathcal{P}} D(ind(\mathbf{x}))}{\sum_{i=1}^{|\mathcal{P}|} D(i)} \\
&= \frac{1}{C}\sum_{i=1}^{|\mathcal{P}|} D(i) - \sum_{\mathbf{x}\in\mathcal{P}} D(ind^{+}(\mathbf{x}) + rank(\mathbf{x}) - 1) \\
&= \frac{1}{C}\sum_{\mathbf{x}\in\mathcal{N}} D(ind^{-}(\mathbf{x}) + rank(\mathbf{x}) - 1) - D(|\mathcal{P}| + ind^{-}(\mathbf{x})) \\
&= \sum_{\mathbf{x}\in\mathcal{N}} \delta_{ind^{-}(\mathbf{x})}(rank(\mathbf{x}))
\end{aligned}$$

as desired. As for (C2) and (C3), let us realize that

$$\delta_j(i+1) - \delta_j(i) = \frac{1}{C}\left(D(i+j) - D(i+j-1)\right).$$

Then (C3) becomes trivial and checking (C2) reduces to

$$D(i+j+1) + D(i+j-1) \geq 2D(i+j)$$

which follows from convexity of the function D. □

Having established that both the AP and the NDCG loss are QS-suitable, the rest of the section will deal with a general QS-suitable loss function. A reader who is interested in employing another loss function need only check whether the required conditions are satisfied in order to use our approach.

3.3.2 Key Observations for QS-Suitable Loss

Before describing our algorithm in detail, we first provide some key observations which enable efficient optimization for QS-suitable loss functions. To this end, let us define an array $\{s_i^+\}_{i=1}^{|\mathcal{P}|}$ of positive sample scores and an array $\{s_i^-\}_{i=1}^{|\mathcal{N}|}$ of negative sample scores. Furthermore, for purely notational purposes, let $\{s_i^*\}$ be the array $\{s_i^-\}$ sorted in descending order. For $j \in \{1, \ldots, |\mathcal{N}|\}$ we denote the index of s_j^- in $\{s_i^*\}$ as j^*.

With the above notation, we describe some key observations regarding QS-suitable loss functions. Their proofs are for most part straightforward generalizations of results that appeared in [105] in the context of the AP loss. Using the interleaving-dependence property of QS-suitable loss functions and structure of the discriminant function as defined in equation (3.1), we can make the following observation.

Observation 1. *There exists an optimal solution $\bar{\mathbf{R}}$ of problem (3.6) that has positive samples appearing in the descending order of their scores s_i^+ and also the negative samples appearing in descending order of their scores s_i^-.*

Proof. Let $\mathbf{R}$ be any optimal solution. We check that $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ increases if we swap two samples $x, y \in \mathcal{P}$ in $\mathbf{R}$ with $ind(\mathbf{x}) < ind(\mathbf{y})$ and $\phi(\mathbf{x}; \mathbf{w}) < \phi(\mathbf{y}; \mathbf{w})$ (it boils down to $ac + bd > ad + bc$ for $a > b \geq 0$ and $c > d \geq 0$). Since similar argument applies for negative samples, we can conclude that $\mathbf{R}$ already has both negative and positive samples sorted decreasingly. Otherwise, one could perform swaps in $\mathbf{R}$ that would increase the value of the objective, a contradiction with the optimality of $\mathbf{R}$. □

Now, in order to find the optimal ranking $\bar{\mathbf{R}}$, it would seem natural to sort the arrays $\{s_i^+\}$ and $\{s_i^-\}$ in descending order and then find the optimal interleaving ranks $rank(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{N}$. However, we are aiming for complexity below $O(|\mathcal{N}| \log |\mathcal{N}|)$, therefore we can not afford to sort the negative scores. On the other hand, since $|\mathcal{P}| << |\mathcal{N}|$, we are allowed to sort the array of positive scores $\{s_i^+\}$. Given the $\pm$-pattern dependency of the QS-suitable loss functions and structure of the discriminant function as defined in equation (3.1), it is actually possible to compute the optimal ranking by enforcing a weaker ordering on $\{s_i^-\}$ which ensures that for every pair of negative samples $\{\mathbf{x_i}, \mathbf{x_j}\}$ with $rank(\mathbf{x}_i) > rank(\mathbf{x}_j)$, $\mathbf{x_i}$ is ranked higher than $\mathbf{x_j}$.

As a result, solving Problem (3.6) reduces to computing the optimal interleaving ranks (or the optimal vector $\mathbf{r}$ as defined in the previous subsection). Note that the value of the objective can be computed efficiently given a vector $\mathbf{r}$ – for example by constructing any ranking $\mathbf{R}$ which respects $\mathbf{r}$. The following observations allow for an efficient computation of the interleaving rank vector $\mathbf{r}$.

Observation 2. *The entire objective function in problem (3.6) inherits properties (C1) and (C2) of QS-suitable loss functions, that is the following holds:*

(a) *There are functions* $f_j\colon \{1,\dots,|\mathcal{P}|+1\} \to \mathbb{R}$ *for* $j = 1,\dots,|\mathcal{N}|$ *such that the objective function in* (6) *can be written as*

$$\sum_{j=1}^{|\mathcal{N}|} f_j(r_j),$$

where r_j *is the interleaving rank of the negative sample* x *with* $ind^{-}(x) = j$.

(b) *The functions* f_j *inherit property (C2). More precisely, for every* $1 \le j < |\mathcal{N}|$ *and* $1 \le i \le |\mathcal{P}|$ *we have*

$$f_{j+1}(i+1) - f_{j+1}(i) \ge f_j(i+1) - f_j(i).$$

(c) *We can compute* $\operatorname{argmax}_{l \le i \le r} f_j(i)$ *in* $O(r-l)$ *time if we are provided access to the sorted array* $\{s_i^+\}$ *and to the score of the negative sample* x *with* $ind^{-}(x) = j$.

Proof. It can be verified with a short computation that the objective function $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ decomposes into contributions of negative and positive samples as follows:

$$\begin{aligned} F(\mathbf{X}, \mathbf{R}; \mathbf{w}) &= \frac{1}{|\mathcal{P}|\,|N|} \sum_{\mathbf{x}\in\mathcal{P}} \sum_{\mathbf{y}\in\mathcal{N}} \mathbf{R}_{\mathbf{x},\mathbf{y}}(\phi(\mathbf{x};\mathbf{w}) - \phi(\mathbf{y};\mathbf{w})) \\ &= \sum_{\mathbf{x}\in\mathcal{P}} c(\mathbf{x})\phi(\mathbf{x};\mathbf{w}) + \sum_{\mathbf{y}\in\mathcal{N}} c(\mathbf{y})\phi(\mathbf{y};\mathbf{w}), \end{aligned}$$

where

$$c(\mathbf{x}) = \frac{|\mathcal{N}| + 2 - 2rank(\mathbf{x})}{|\mathcal{P}|\,|\mathcal{N}|},\ c(\mathbf{y}) = \frac{|\mathcal{P}| + 2 - 2rank(\mathbf{y})}{|\mathcal{P}|\,|\mathcal{N}|}.$$

In particular, assuming already that $\{s_i^+\}$ is sorted, and that $\mathbf{R}$ is induced by a vector of interleaving ranks $\mathbf{r}$, one has

$$F(\mathbf{X}, \mathbf{R}; \mathbf{w}) = \sum_{i=1}^{|\mathcal{P}|} c_i^+ s_i^+ + \sum_{j=1}^{|\mathcal{N}|} c_j^- s_j^*,$$

where

$$c_i^+ = \frac{|\mathcal{N}| + 2 - 2r_i^+}{|\mathcal{P}|\,|\mathcal{N}|}, \qquad c_j^- = \frac{|\mathcal{P}| + 2 - 2r_j}{|\mathcal{P}|\,|\mathcal{N}|}.$$

Here, r_i^+ stands for the interleaving rank of the i-th positive sample, which can be computed as $r_i^+ = 1 + |\{j : r_j \le i\}|$.

We can slightly modify the above decomposition in order to incorporate the array $\{s_i^+\}$:

$$\begin{aligned}
&F(\mathbf{X},\mathbf{R};\mathbf{w})\\
&=\sum_{\mathbf{y}\in\mathcal{N}}\left(c(\mathbf{y})\phi(\mathbf{y};\mathbf{w})+\sum_{\mathbf{x}\in\mathcal{P}}\mathbf{R}_{\mathbf{x},\mathbf{y}}\phi(\mathbf{x};\mathbf{w})\right)\\
&=\frac{1}{|\mathcal{N}|\,|\mathcal{P}|}\sum_{j=1}^{|\mathcal{N}|}\left((|\mathcal{P}|+2-2r_j)s_j^*+2\sum_{i=1}^{r_j-1}s_i^+-\sum_{i=1}^{|\mathcal{P}|}s_i^+\right)
\end{aligned}$$

This, in combination with (C1), defines the functions f_j for $j=1,\ldots,|\mathcal{N}|$ as

$$f_j=\frac{1}{|\mathcal{N}|\,|\mathcal{P}|}\left((|\mathcal{P}|+2-2r_j)s_j^*+2\sum_{i=1}^{r_j-1}s_i^+-\sum_{i=1}^{|\mathcal{P}|}s_i^+\right). \tag{3.10}$$

As for the condition (C2), we have

$$f_j(i+1)-f_j(i)=\frac{2(s_i^+-s_j^*)}{|\mathcal{N}|\,|\mathcal{P}|}+\delta_j(i+1)-\delta_j(i),$$

where, let us be reminded, $\{s_j^*\}$ is the sorted array of scores of negative samples. After writing analogous equality for $j+1$ and using that (C2) holds for functions δ_j, we can check that the desired inequality

$$f_{j+1}(i+1)-f_{j+1}(i)\geq f_j(i+1)-f_j(i)$$

follows from $s_{j+1}^*\leq s_j^*$.

Note that for computing the $\operatorname{argmax} f_j(i)$ it is sufficient to compute all discrete derivatives (i.e. all differences $f_j(i+1)-f_j(i)$); the actual values of f_j are in fact not needed. For δ_j we know that one such evaluation is constant time and it is also the same for f_j since we assumed to have access to s_j^*. □

Let opt_i be the optimal interleaving rank for the negative sample with the i^{th} rank in the sorted list $\{s_i^*\}$ and $\mathbf{opt}=\{opt_i|j=1,\ldots,|\mathcal{N}|\}$ be the optimal interleaving rank vector. The above observation gives us the opportunity to compute the interleaving rank for each negative sample independently. This is however not obvious. One certainly can maximize each f_j but the resulting vector $\mathbf{r}$ may not induce any ranking – its entries may not be monotone. But as a matter of fact, this does not happen and the following observation gives the precise guarantee.

Observation 3. *If $i<j$, then $opt_i\leq opt_j$.*

Proof. Recall that opt_j is the highest rank with maximal value of the corresponding $f_{j'}$. It suffices to prove that for $i_{j+1}=\max\operatorname{argmax} f_{j+1}$ and $i_j=\max\operatorname{argmax} f_j$, we have $i_{j+1}\geq i_j$. Since by

Observation 2 functions f_j inherit property (C2), we can compare the discrete derivatives of f_j and f_{j+1}, all left to do is to formalize the discrete analogue of what seems intuitive for continuous functions.

Assume $i_{j+1} < i_j$. Then since

$$\begin{aligned} f_{j+1}(i_j) - f_{j+1}(i_{j+1}) &= \sum_{i=i_{j+1}}^{i_j-1} f_{j+1}(i+1) - f_{j+1}(i) \\ &\geq \sum_{i=i_{j+1}}^{i_j-1} f_j(i+1) - f_j(i) \\ &= f_j(i_j) - f_j(i_{j+1}) \geq 0, \end{aligned}$$

we obtain that $i_j \in \operatorname{argmax} f_{j+1}$ and as $i_j > i_{j+1} = \max \operatorname{argmax} f_{j+1}$, we reach the expected contradiction. $\square$

All in all, it suffices to compute the vector $\mathbf{opt}$ in which $opt_j = \max \operatorname{argmax} f_j$ (the maximum ensures that ties are broken consistently). However, we actually need not do this computation for all the $|\mathcal{N}|$ negative samples. This is because, since the interleaving rank for any negative sample can only belong to $[1, |\mathcal{P}| + 1]$ and $|\mathcal{P}| << |\mathcal{N}|$, many of the negative samples would have the same interleaving rank. This fact can be leveraged to improve the efficiency of the algorithm for finding $\mathbf{opt}$ by making use of the above observation. Knowing that $opt_i = opt_j$ for some $i < j$, we can conclude that $opt_i = opt_k = opt_j$ for each $i < k < j$. This provides a cheap way to compute some parts of the vector $\mathbf{opt}$ if an appropriate sequence is followed for computing the interleaving ranks. Even without access to the fully sorted set $\{s^*_j\}$, we can still find s^*_j, the j-highest element in $\{s^-_i\}$, for a fixed j, in $O(|\mathcal{N}|)$ time. This would lead to an $O(|\mathcal{P}|\,|\mathcal{N}|)$ algorithm but we may at each step modify $\{s^-_i\}$ slowly introducing the correct order. This will make the future searches for s^*_j more efficient.

3.3.3 Divide and Conquer

Algorithm 4 describes the main steps of our approach. Briefly, we begin by detecting $s^*_{|\mathcal{N}|/2}$ that is the median score among the negative samples. We use this to compute $opt_{|\mathcal{N}|/2}$. Given $opt_{|\mathcal{N}|/2}$, we know that for all $j < |\mathcal{N}|/2$, $opt_j \in [1, opt_{|\mathcal{N}|/2}]$ and for all $j > |\mathcal{N}|/2$, $opt_j \in [opt_{|\mathcal{N}|/2}, |\mathcal{P}| + 1]$. This observation allows us to employ a divide-and-conquer recursive approach.

In more detail, we use two classical linear time array manipulating procedures MEDIAN and SELECT. The first one outputs the index of the median element. The second one takes as its input an index of a particular element x. It rearranges the array such that x separates higher-ranked elements from lower-ranked elements (in some total order). For example, if array s^- contains six scores $[a\ b\ 4.5\ 6\ 1\ c]$ then `Median`(3, 5) would return 3 (the index of score 4.5), while calling `Select`(3, 3, 5) would rearrange the array to $[a\ b\ 1\ 4.5\ 6\ c]$ and return 4 (the new index of 4.5). The SELECT procedure is a subroutine of the classical QUICKSORT algorithm.

Using the two aforementioned procedures in conjunction with the divide-and-conquer strategy allows us to compute the entire interleaving rank vector **opt** and this in turn allows us to compute the semi-gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$, as in equation (3.5), efficiently.

Algorithm 4 Recursive procedure for finding all interleaving ranks.

Description: The function finds optimal interleaving rank for all $neg[i]$ with $i \in [\ell^-, r^-]$ given that,

(i) array s^- is partially sorted, namely

$\texttt{MAX}(s^-[1 \dots \ell^- - 1]) \leq \texttt{MIN}(s^-[\ell^- \dots r^-])$

$\texttt{MAX}(s^-[\ell^- \dots r^-]) \leq \texttt{MIN}(s^-[r^- + 1 \dots |\mathcal{N}|])$

(ii) optimal interleaving ranks for $i \in [\ell^-, r^-]$ lie in the interval $[\ell^+, r^+]$.

```
1: function OptRanks(int $\ell^-$, int $r^-$, int $\ell^+$, int $r^+$)
2:     if $\ell^+ = r^+$ then
3:         set $opt_i = \ell^+$ for each $i \in [\ell^-, r^-]$ and return
4:     $m$ = Median($\ell^-$, $r^-$)  ▷ gives the index of the median score in a subarray of $s^-$
5:     $m$ = Select($m$, $\ell^-$, $r^-$)  ▷ splits the subarray by $s = s^-[m]$, returns the new index of $s$
6:     Find $opt_m$ by trying all options in $[\ell^+, r^+]$
7:     if $\ell^- < m$ then OptRanks($\ell^-$, $m$−1, $\ell^+$, $opt_m$)
8:     if $m < r^-$ thenOptRanks($m$+1, $r^-$, $opt_m$, $r^+$)
```

Figure 3.1 provides an illustrative example of our divide-and-conquer strategy. Here, $|\mathcal{N}| = 11$ and $|\mathcal{P}| = 2$. We assume that the optimal interleaving rank vector **opt** is $[1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3]$. Let us now go through the procedure in which Algorithm 4 computes this optimal interleaving rank vector. Before starting the recursive procedure, we only sort the positive samples according to their scores and do not sort the negative samples. To start with, we call `OptRanks`$(1, 11, 1, 3)$. We find the negative sample with the median score (6th highest in this case) and compute its optimal interleaving rank opt_6 to be 2. In the next step of the recursion, we make the following calls: `OptRanks`$(1, 5, 1, 2)$ and `OptRanks`$(7, 11, 2, 3)$. These calls compute opt_3 and opt_9 to be 2. In the next set of recursion

_	_	_	_	_	_	_	_	_	_	_
_	_	_	_	_	$\underline{[2]}$	_	_	_	_	_
_	_	$\underline{[2]}$	_	_	$\underline{[2]}$	_	_	$\underline{[2]}$	_	_
_	$\underline{[2]}$	$\underline{[2]}$	$\underline{2}$	$\underline{2}$	$\underline{[2]}$	$\underline{2}$	$\underline{2}$	$\underline{[2]}$	$\underline{[3]}$	_
$\underline{[1]}$	$\underline{[2]}$	$\underline{[2]}$	$\underline{2}$	$\underline{2}$	$\underline{[2]}$	$\underline{2}$	$\underline{2}$	$\underline{[2]}$	$\underline{[3]}$	$\underline{3}$

Figure 3.1: *Example illustrating the path followed by the quick sort flavored recursive algorithm while computing the interleaving rank vector* $\mathbf{opt}$. *Row correspond to the status of* $\mathbf{opt}$ *at selected time steps.*

calls however, the calls `OptRanks`$(4, 5, 2, 2)$ and `OptRanks`$(7, 8, 2, 2)$, get terminated in step 4 of Algorithm 4 and opt_j for $j = 4, 5, 7, 8$ are assigned without any additional computation. We then continue this procedure recursively for progressively smaller intervals as described in Algorithm 4. Leveraging the fact stated in observation 3, our algorithm has to explicitly compute the interleaving rank for only 6 (shown in square brackets) out of the 11 negative samples. In a typical real data set, which is skewed more in favor of the negative samples, the expected number of negative samples for which the interleaving rank has to be explicitly computed is far less than $|\mathcal{N}|$. In contrast, the algorithm proposed by Yue *et al.* in [105] first sorts the entire negative set in descending order of their scores and explicitly computes the interleaving rank for each of the $|\mathcal{N}|$ negative samples.

3.3.4 Computational Complexity

The computational complexity of the divide-and-conquer strategy to estimate the output of problem (3.6), is given by the following theorem.

Theorem 7. *If Δ is* QS*-suitable, then the task* (3.6) *can be solved in time* $O(|\mathcal{N}|\log|\mathcal{P}| + |\mathcal{P}|\log|\mathcal{P}| + |\mathcal{P}|\log|\mathcal{N}|)$*, which in the most common case* $|\mathcal{N}| > |\mathcal{P}|$ *reduces to* $O(|\mathcal{N}|\log|\mathcal{P}|)$ *and any comparison-based algorithm would require* $\Omega(|\mathcal{N}|\log|\mathcal{P}|)$ *operations.*

Outside running Algorithm 1, the entire computation also consists of preprocessing (sorting positive samples by their scores) and post processing (computing the output from vector $\mathbf{opt}$). These subroutines have only one non-linear complexity term – $O(|\mathcal{P}|\log|\mathcal{P}|)$ coming from the sorting. Therefore, it remains to establish the complexity of Algorithm 1 as $O(|\mathcal{N}|\log|\mathcal{P}| + |\mathcal{P}|\log|\mathcal{N}|)$.

To this end, let us denote $n = r^- - \ell^- + 1$ and $p = r^+ - \ell^+ + 1$, and set $T_{neg}(n,p)$, $T_{pos}(n,p)$ as the total time spent traversing the arrays of negative and positive sample scores, respectively, including recursive calls. The negative score array is traversed in the MEDIAN and SELECT procedures and the positive scores are traversed when searching for opt_m. The latter has by complexity $O(p)$, due to Observation 2(c), whose assumption are always satisfied during the run of the algorithm.

Proposition 8. *The runtimes $T_{neg}(n,p)$ and $T_{pos}(n,p)$ satisfy the following recursive inequalities*

$$\begin{aligned}
T_{neg}(n,p) &\le Cn + T_{neg}(n/2,p_1) + T_{neg}(n/2,p_2) \\
&\qquad\qquad\qquad \textit{for some} \quad p_1 + p_2 = p+1, \\
T_{pos}(n,p) &\le Cp + T_{pos}(n/2,p_1) + T_{pos}(n/2,p_2) \\
&\qquad\qquad\qquad \textit{for some} \quad p_1 + p_2 = p+1, \\
T_{neg}(n,1) &\le Cn, \qquad T_{neg}(1,p) = 0, \\
T_{pos}(n,1) &= 0, \qquad T_{pos}(1,p) \le Cp
\end{aligned}$$

for a suitable constant C. These inequalities imply $T_{neg}(n,p) \le C'n\log(1+p)$ and $T_{pos}(n,p) \le C'(p-1)\log(1+n)$ for another constant C'. Thus the running time of Algorithm 1, where $p = |\mathcal{P}|+1$, $n = |\mathcal{N}|$, is $O(|\mathcal{N}|\log|\mathcal{P}| + |\mathcal{P}|\log|\mathcal{N}|)$.

Proof. The recursive inequalities follow from inspection of Algorithm 1. As for the “aggregated” inequalities, we proceed in both cases by induction. For the first inequality the base step is trivial for high enough constant C' and for the inductive step we may write

$$\begin{aligned}
T_{neg}(n,p) &\le Cn + T_{neg}(n/2,p_1) + T_{neg}(n/2,p_2) \\
&\le Cn + \frac{1}{2}C'n\log(1+p_1) + \frac{1}{2}C'n\log(1+p_2) \\
&= C'n\left(\frac{C}{C'} + \log\sqrt{(1+p_1)(1+p_2)}\right) \\
&\le C'n\log(p_1+p_2) = C'n\log(1+p)
\end{aligned}$$

where in the last inequality we used that

$$1 + (1+p_1)(1+p_2) \le (p_1+p_2)^2$$

for integers p_1, p_2 with $p_1 + p_2 = p + 1 \geq 3$. That makes the last inequality true for sufficiently high C' (not depending on n and p).

The proof of the second inequality is an easier variation on the previous technique. $\square$

3.3.5 Lower Bound on Complexity

In order to prove the matching lower bound (among comparison-based algorithms), we intend to use the classical information theoretical argument: There are many possible outputs and from each comparison we receive one bit of information, therefore we need "many" comparison to shatter all output options.

Proposition 9. *Let Δ be a loss function. Then any comparison-based algorithm for Problem* (6) *requires* $\Omega(|\mathcal{N}| \log |\mathcal{P}|)$ *operations.*

Proof. Since the negative samples are unsorted on the input and the scores are arbitrary, every possible mapping from $\{1, \ldots, |\mathcal{N}|\}$ to $\{1, \ldots, |\mathcal{P}| + 1\}$ may induce the (unique) optimal assignment of interleaving ranks. There are $(|\mathcal{P}| + 1)^{|\mathcal{N}|}$ possibilities to be distinguished and each comparison has only two possible outcomes. Therefore we need $\log_2 \left((|\mathcal{P}| + 1)^{|\mathcal{N}|} \right) \in \Omega(|\mathcal{N}| \log |\mathcal{P}|)$ operations. $\square$

Note that the above theorem not only establishes the superior runtime of our approach ($O(|\mathcal{N}| \log |\mathcal{P}|)$ compared to $O(|\mathcal{N}| \log |\mathcal{N}|)$ of [105]), it also provides an asymptotic lower bound for comparison based algorithms. However, it does not rule out the possibility of improving the constants hidden within the asymptotic notation for a given loss function. In the next section we present a method that exploits the additional structure of the AP loss to further speed-up our algorithm.

3.4 Efficient Optimization for AP loss

In this section, we propose a method for further speeding up the optimization procedure of AP loss. In order to find the most violated ranking as denoted in problem 3.6, the quicksort flavoured algorithm presented in 4 iteratively assigns the optimal interleaving rank $opt_j \in \{1, \cdots, |\mathcal{P}| + 1\}$ for all negative samples $\mathbf{x}_j$'s without having to do explicit computation for each of them. The interleaving rank opt_j specifies that the negative sample $\mathbf{x}_j$ must be ranked between the $(opt_j - 1)$-th and the opt_j-th positive sample. The computation of the optimal interleaving rank for a particular negative sample requires us to maximize the discrete function $f_j(i)$ over the domain $i \in \{1, \cdots, |\mathcal{P}|\}$. Like Yue *et al.* [105], algorithm 4 uses a simple linear algorithm for this step, which takes $O(|\mathcal{P}|)$ time. In contrast, we propose a more efficient algorithm to maximize $f_j(\cdot)$, which exploits the special structure of this discrete function.

Before we describe our efficient algorithm in detail, we require the definition of a unimodal function. A discrete function $f : \{1, \cdots, p\} \leftarrow \mathbb{R}$ is said to be unimodal if and only if there exists a $k \in \{1, \cdots, p\}$ such that

$$\begin{aligned} &f(i) \leq f(i+1), \forall i \in \{1, \cdots, k-1\}, \\ &f(i-1) \geq f(i), \forall i \in \{k+1, \cdots, p\}. \end{aligned} \tag{3.11}$$

In other words, a unimodal discrete function is monotonically non-decreasing in the interval $[1, k]$ and monotonically non-increasing in the interval $[k, p]$. The maximization of a unimodal discrete function over its domain $\{1, \cdots, p\}$ simply requires us to find the index k that satisfies the above properties. The maximization can be performed efficiently, in $O(\log(p))$ time, using binary search.

We are now ready to state the main result that allows us to compute the optimal interleaving rank of a negative sample efficiently.

Proposition 10. *The discrete function $\delta_j(i)$, defined in equation (3.9), is unimodal in the domain $\{1, \cdots, p\}$, where $p = \min\{|\mathcal{P}|, j\}$.*

We provide a proof for the above proposition. Before moving to the proposition, we first state the following lemmas, which easily lead to the proposition. For the sake of clarity of discussion, we will split the summand term in the summation $\delta_j(i)$ as follows:

$$\begin{aligned} \delta_j(i) &= f_1(j,i) + f_2(j,i) = \sum_{k=i}^{|\mathcal{P}|} g_1(j,k) + \sum_{k=i}^{|\mathcal{P}|} g_2(j,k), \\ g_1(j,k) &= \frac{1}{|\mathcal{P}|}\left(\frac{j}{j+k} - \frac{j-1}{j+k-1}\right), g_2(j,k) = -\frac{2\left(s_k^p - s_j^n\right)}{|\mathcal{P}||\mathcal{N}|}. \end{aligned}$$

Please note that the functions $f_1(j,i)$ and $f_2(j,i)$ are cumulative sums of $g_1(j,i)$ and $g_2(j,i)$ respectively, in the decreasing direction of i. Therefore, for ease of reasoning, we shall analyse the trend of these functions in the decreasing direction of i.

Lemma 11. *For $k < j$, $g_1(j,k)$ monotonically decreases with decreasing k, that is $\forall\ k < j$*
$g_1(j, k-1) \leq g_1(j,k)$.

Proof. For $j \geq 1$ and $k \geq 1$, $(j+k) > j \;\Rightarrow\; j(j+k) - (j+k) \;<\; j(j+k) - j \;\Rightarrow\; \frac{j}{j+k} > \frac{j-1}{j+k-1}$. So, term $g_1(j,k) > 0$ for all $k \geq 1$. It can also be verified that the function $g_1(j,k)$ is 0 at 0 and has a single maxima for $k \in \Re^+$, at $k = \sqrt{j(j-1)}$. From this we can conclude that for discrete $k \in \mathbb{Z}^+$, $g_1(j,k)$ would have maximum value either at $k = j$ or $k = j-1$. Therefore, for $k < j$, $g_1(j,k)$ would monotonically decrease with decreasing k. □

Lemma 12. *For $k < j$, $g_2(j,k)$ monotonically decreases with decreasing k, that is $\forall\; k < j$ $g_2(j,k-1) \leq\; g_2(j,k)$.*

Proof. In $g_2(j,k)$, the negative score s^n_j is a constant for a given j. Whereas, the positive scores s^p_k being sorted in descending order, monotonically increase as k decreases. Therefore, $g_2(j,k)$ which is $-s^p_k + constant$, monotonically decreases as k decreases. □

Proposition 13. *The discrete function $\delta_j(i)$, defined in equation-5 of the main text, is unimodal in the domain $\{1, \cdots, p\}$, where $p = \min\{|\mathcal{P}|, j\}$.*

Proof. From lemmas 11 and 12, for $k < j$, $g_1(j,k)$ and $g_2(j,k)$ monotonically decreases with decreasing k. As a result, $g_1(j,k) + g_2(j,k)$ also monotonically decreases when k is decreased from right to left of the number line. Here, there can be 3 scenarios,

(i) $(g_1(j,1) + g_2(j,1)) \geq 0$. In this case, as the function is monotonic and decreases towards left,

$$\begin{aligned} & (g_1(j,i) + g_2(j,i)) \geq 0, for\; i \in \{1,2,...,j\} \\ \Rightarrow \quad & \delta_j(i) - \delta_j(i+1) \geq 0, for\;\; i \in \{1,2,...,\} \\ \Rightarrow \quad & \delta_j(i) \geq \delta_j(i+1), for\;\; i \in \{1,2,...,\} \end{aligned}$$

Therefore, according to definition of unimodality, $\delta_j(i)$ would be unimodal with $k = 1$.

(ii) $(g_1(j,j-1) + g_2(j,j-1)) \leq 0$. In this case, using similar reasoning as above,

$$\begin{aligned} & (g_1(j,i) + g_2(j,i)) \leq 0, for\; i \in \{j-1,...,1\} \\ \Rightarrow \quad & \delta_j(i) - \delta_j(i+1) \leq 0, for\;\; i \in \{j-1,...,1\} \\ \Rightarrow \quad & \delta_j(i) \leq \delta_j(i+1), for\;\; i \in \{j-1,...,1\} \end{aligned}$$

Therefore, $\delta_j(i)$ would be unimodal with $k = j - 1$.

(iii) $(g_1(j,1) + g_2(j,1)) \leq 0$ and $(g_1(j,j-1) + g_2(j,j-1)) \geq 0$. In this case, there should exist a point across which the function $(g_1 + g_2)$ changes its sign from positive to negative when moving from right to left. In other words, there should exist $k \in 1, 2, \ldots, j-1$, such that,

$$\begin{aligned} & (g_1(j,i) + g_2(j,i)) \geq 0, i \in \{k+1, ..., j\} \\ & (g_1(j,i) + g_2(j,i)) \leq 0, i \in \{1, ..., k\} \\ \Rightarrow \quad & \delta_j(i) - \delta_j(i+1) \geq 0, for \;\; i \in \{k, ..., j-1\} \\ & \delta_j(i) - \delta_j(i+1) \leq 0, for \;\; i \in \{j-1, ..., 1\} \\ \Rightarrow \quad & \delta_j(i) \geq \delta_j(i+1), for \;\; i \in \{k, ..., j-1\} \\ & \delta_j(i) \leq \delta_j(i+1), for \;\; i \in \{j-1, ..., 1\} \end{aligned}$$

Here too, $\delta_j(i)$ satisfies the conditions for unimodality with k being the maximum point.

In all the 3 of the exhaustive cases, $\delta_j(i)$ satisfies the conditions for unimodality. Hence, $\delta_j(i)$ is unimodal in the region $\{1, 2, \ldots, j-1\}$. As a function which is unimodal in a certain region would also be unimodal in a subset of the region, $\delta_j(i)$ is unimodal in the region $\{1, 2, \ldots, p\}$, where, $p = \min(|\mathcal{P}|, j)$. □

Algorithm 5 *Efficient search for the optimal interleaving rank of a negative sample.*

```
   Input: $\{\delta_j(i), i = 1, \cdots, |\mathcal{P}|\}$.
1: $p = \min\{|\mathcal{P}|, j\}$.
2: Compute an interleaving rank $i_1$ as
       $$i_i = \underset{i \in \{1, \cdots, p\}}{\text{argmax}} \; \delta_j(i). \quad (3.12)$$
3: Compute an interleaving rank $i_2$ as
       $$i_2 = \underset{i \in \{p+1, \cdots, |\mathcal{P}|\}}{\text{argmax}} \; \delta_j(i). \quad (3.13)$$
4: Compute the optimal interleaving rank $opt_j$ as
       $$opt_j = \begin{cases} i_1 & \text{if } \delta_j(i_1) \geq \delta_j(i_2), \\ i_2 & \text{otherwise.} \end{cases} \quad (3.14)$$
```

Using the above proposition, the discrete function $\delta_j(i)$ can be optimized over the domain $\{1, \cdots, |\mathcal{P}|\}$ efficiently as described in Algorithm 5. Briefly, our efficient search algorithm finds an interleaving ranking i_1 over the domain $\{1, \cdots, p\}$, where p is set to $\min\{|\mathcal{P}|, j\}$ in order to ensure that the function $\delta_j(\cdot)$ is unimodal (step 2 of Algorithm 5). Since i_1 can be computed using binary search, the computational complexity of this step is $O(\log(p))$. Furthermore, we find an interleaving ranking i_2 over the domain $\{p+1, \cdots, |\mathcal{P}|\}$ (step 3 of Algorithm 5). Since i_2 needs to be computed using linear search, the computational complexity of this step is $O(|\mathcal{P}| - p)$ when $p < |\mathcal{P}|$ and 0 otherwise. The optimal interleaving ranking opt_j of the negative sample $\mathbf{x}_j$ can then be computed by comparing the values of $\delta_j(i_1)$ and $\delta_j(i_2)$ (step 4 of Algorithm 5).

Note that, in a typical training dataset, the negative samples significantly outnumber the positive samples, that is, $|\mathcal{N}| \gg |\mathcal{P}|$. For all the negative samples $\mathbf{x}_j$ where $j \geq |\mathcal{P}|$, p will be equal to $|\mathcal{P}|$. Hence, the maximization of $\delta_j(\cdot)$ can be performed efficiently over the entire domain $\{1, \cdots, |\mathcal{P}|\}$ using binary search in $O(\log(|\mathcal{P}|))$ as opposed to the $O(|\mathcal{P}|)$ time suggested in [105].

3.5 Experiments

We demonstrate the efficacy of our approach on three vision tasks with increasing level of complexity. First, we use the simple experimental setup of doing action classification on the PASCAL VOC 2011 data set using a shallow model. This experimental set up allows us to thoroughly analyze the performance of our method as well as the baselines by varying the sample set sizes. Second, we apply our method to a large scale experiment of doing object detection on the PASCAL VOC 2007 data set using a shallow model. This demonstrates that our approach can be used in conjunction with a large data set consisting of millions of samples. Finally, we demonstrate the effectiveness of our method for layer wise training of a deep network on the task of image classification using the CIFAR-10 data set.

3.5.1 Action Classification

Data set. We use the PASCAL VOC 2011 [30] action classification data set for our experiments. This data set consists of 4846 images, which include 10 different action classes. The data set is divided into two parts: 3347 'trainval' person bounding boxes and 3363 'test' person bounding boxes. We use the 'trainval' bounding boxes for training since their ground-truth action classes are known. We evaluate the accuracy of the different models on the 'test' bounding boxes using the PASCAL evaluation server.

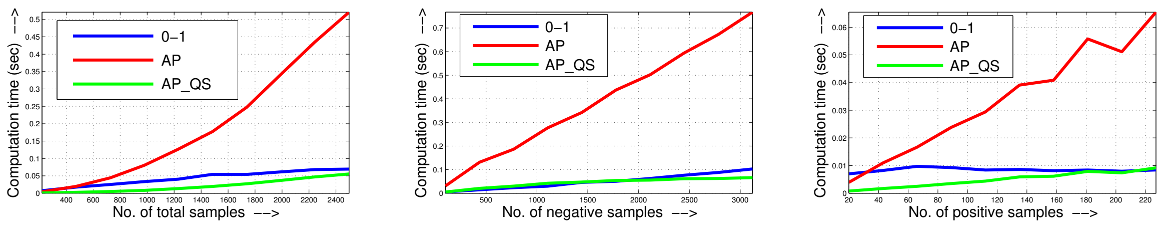


Figure 3.2: *Total computation time for multiple calls to loss augmented inference during model training, when the number of total, negative and positive samples are varied. Here, 0-1,* AP *and* AP_QS *correspond to loss augmented inference procedures for 0-1 loss, for* AP *loss using [105] and for* AP *loss using our method respectively. It can be seen that our method scales really well with respect to sample set sizes and takes computational time that is comparable to what is required for simpler 0-1 decomposable loss.*

Model. We use structured SVM models as discriminant functions and use the standard poselet [63] activation features to define the sample feature for each person bounding box. The feature vector consists of 2400 action poselet activations and 4 object detection scores. We refer the reader to [63] for details regarding the feature vector.

Methods. We show the effectiveness of our method in optimizing both AP loss and NDCG loss to learn the model parameters. Specifically, we report the computational time for the loss-augmented inference evaluations. For AP loss, we compare our method (referred to as AP_QS) with the loss-augmented inference procedure described in [105] (referred to as AP). We also report results with the additional speedup provided by the method proposed in section 3.4 (referred to as AP_QS_ES). For NDCG loss, we compare our method (referred to as NDCG_QS) with the loss-augmented inference procedure described in [14] (referred to as NDCG). We also report results for loss-augmented inference evaluations when using the simple decomposable 0-1 loss function (referred to as 0-1). The hyperparameters involved are fixed using 5-fold cross-validation on the 'trainval' set.

0-1	AP	AP_QS	AP_QS_ES	NDCG	NDCG_QS
0.0694	0.7154	0.0625	0.0609	6.8019	0.0473

Table 3.1: *Total computation time (in seconds) when using the different methods, for multiple calls to loss augmented inference during model training. The reported time is averaged over the training for all the action classes.*

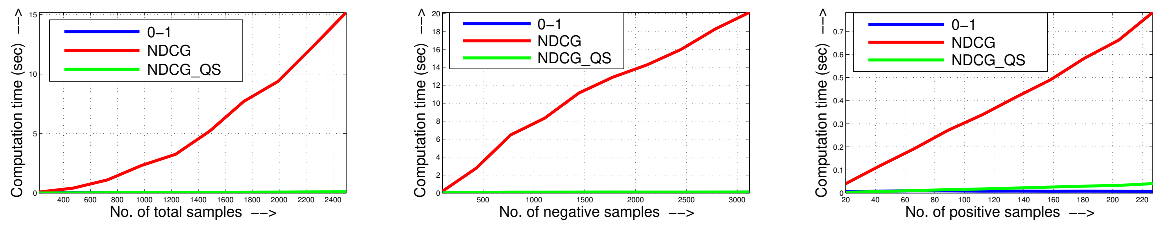


Figure 3.3: *Total computation time for multiple calls to loss augmented inference during model training, when the number of total, negative and positive samples are varied. Here, 0-1,* NDCG *and* NDCG_QS *correspond to loss augmented inference procedures for 0-1 loss, for* NDCG *loss using [14] and for* NDCG *loss using our method respectively. As can be seen, our approach scales elegantly with respect to sample set sizes and is comparable to the simpler 0-1 decomposable loss in terms of computation time.*

Results. When we minimize AP loss on the training set to learn the model parameters, we get a mean AP of 51.196 on the test set. In comparison, minimizing 0-1 loss to learn model parameters leads to a mean AP value of 47.934 on the test set. Similarly, minimizing NDCG loss for parameter learning gives a superior mean NDCG value of 85.521 on the test set, compared to that of 84.3823 when using 0-1 loss. The AP and NDCG values obtained on the test set for individual action classes can be found in the appendix. This clearly demonstrates the usefulness of directly using rank based loss functions like AP loss and NDCG loss for learning model parameters, instead of using simple decomposable loss functions like 0-1 loss as surrogates.

The time required for the loss augmented inference evaluations, while optimizing the different loss functions for learning model parameters, are shown in Table 3.1. It can be seen that using our method (AP_QS, NDCG_QS) leads to reduction in computational time by a factor of more than 10, when compared to the methods proposed in [105] and [14] for AP loss and NDCG loss respectively. It can also be observed that although the computational time for each call to loss-augmented inference for 0-1 loss is

0-1	AP	AP_QS	AP_QS_ES	NDCG	NDCG_QS
0.48±0.03	16.29±0.18	1.48±0.39	1.41±0.22	71.07±1.57	0.55±0.11

Table 3.2: *Mean computation time (in milliseconds) when using the different methods, for single call to loss augmented inference. The reported time is averaged over all training iterations and over all the action classes.*

slightly less than that for AP loss and NDCG loss (Table 3.2), in some cases we observe that we required more calls to optimize the 0-1 loss. As a result, in those cases training using 0-1 loss is slower than training using AP or NDCG loss with our proposed method. We also report results of further speedup for AP loss, obtained with the method proposed in section 3.4 (AP_QS_ES). As can be seen in Table 3.1 and Table 3.2, the proposed method does provide a significant empirical speedup.

In order to understand the effect of the size and composition of the data set on our approaches, we perform 3 experiments with variable number of samples for the action class 'phoning'. First, we vary the total number of samples while fixing the positive to negative ratio to $1 : 10$. Second, we vary the number of negative samples while fixing the number of positive samples to 227. Third, we vary the number of positive samples while fixing the number of negative samples to 200. As can be seen in Fig. 3.2 and Fig. 3.3, the time required for loss-augmented inference is significantly lower using our approach for both AP and NDCG loss.

3.5.2 Object Detection

Data set. We use the PASCAL VOC 2007 [30] object detection data set, which consists of a total of 9963 images. The data set is divided into a 'trainval' set of 5011 images and a 'test' set of 4952 images. All the images are labeled to indicate the presence or absence of the instances of 20 different object categories. In addition, we are also provided with tight bounding boxes around the object instances, which we ignore during training and testing. Instead, we treat the location of the objects as a latent variable. In order to reduce the latent variable space, we use the selective-search algorithm [95] in its fast mode, which generates an average of 2000 candidate windows per image. This results in a training set size of approximately 10 million bounding boxes.

Model. For each candidate window, we use a feature representation that is extracted from a trained Convolutional Neural Network (CNN). Specifically, we pass the image as input to the CNN and use the activation vector of the penultimate layer of the CNN as the feature vector. Inspired by the R-CNN pipeline of Girshick *et al.* [36], we use the CNN that is trained on the ImageNet data set [22], by rescaling each candidate window to a fixed size of 224×224. The length of the resulting feature vector is 4096. However, in contrast to [36], we do not assume ground-truth bounding boxes to be available for training images. We instead optimize AP loss in a weakly supervised framework to learn the parameters of the SVM based object detectors for the 20 object categories.

Methods. We use our approach to learn the parameters of latent AP-SVMs [4] for each object category. In our experiments, we fix the hyperparameters using 5-fold cross-validation. During testing, we evaluate each candidate window generated by selective search and use non-maxima suppression to prune highly overlapping detections.

Results. For this task of weakly supervised object detection, using AP loss for learning model parameters leads to a mean test AP of 36.616 which is significantly better than the 29.4995 obtained using 0-1 loss. The AP values obtained on the test set by the detectors for each object class can be found in the appendix. These results establish the usefulness of optimizing AP loss for learning the object detectors. On the other hand, optimizing AP loss for this task places high computational demands due to the size of the data set (5011 'trainval' images) as well as the latent space (2000 candidate windows per image) amounting to around 10 million bounding boxes. We show that using our method for loss-augmented inference (LAI) leads to significant saving in computational time. During training, the total time taken for LAI, averaged over all the 20 classes, was 0.5214 sec for our method which is an order of magnitude better than the 7.623 sec taken by the algorithm proposed in [105]. Thus, using our efficient quicksort flavored algorithm can be critical when optimizing non-decomposable loss functions like AP loss for large scale data sets.

3.5.3 Image Classification

Data set. We use the CIFAR-10 data set [57], which consists of a total of 60,000 images of size 32×32 pixels. Each image belongs to one of 10 specified classes. The data set is divided into a 'trainval' set of 50,000 images and a 'test' set of 10,000 images. From the 50,000 'trainval' images, we use 45,000 for training and 5,000 for validation. For our experiments, all the images are centered and normalized.

Model. We use a deep neural network as our classification model. Specifically, we use a piecewise linear convolutional neural network (PL-CNN) as proposed in [8]. We follow the same framework as [8] for experiments on the CIFAR-10 data set and use a PL-CNN architecture comprising 6 convolutional layers and an SVM last layer. For all our experiments, we use a network that is pre-trained using softmax and cross-entropy loss.

Methods. We learn the weights of the PL-CNN by optimizing AP loss and NDCG loss for the training data set. For comparison, we also report results for parameter learning using the simple decomposable 0-1 loss. We use the layerwise optimization algorithm called LW-SVM, proposed in [8], for optimizing the different loss functions with respect to the network weights. Following the training regime used in [8], we warm start the optimization with a few epochs of Adadelta [106] before running the layer wise optimization. The LW-SVM algorithm involves solving a structured SVM problem for one layer at a time. This requires tens of thousands of calls to loss augmented inference and having an efficient procedure is therefore critical for scalability. We compare our method for loss-augmented inference with the methods described in [105] and [14], for AP loss and NDCG loss respectively.

Results. We get a better mean AP of 85.28 on the test set when we directly optimize AP loss for learning network weights compared to that of 84.22 for 0-1 loss. Similarly, directly optimizing NDCG loss leads to a better mean NDCG of 96.14 on the test set compared to 95.31 for 0-1 loss. This establishes the usefulness of optimizing non-decomposable loss functions like the AP loss and NDCG loss. The LW-SVM algorithm involves very high number of calls to the loss augmented inference procedure. In light of this, the efficient method for loss augmented inference proposed in this work leads to significant reduction in total training time. When optimizing the AP loss, using our method leads to a total training time of 1.589 hrs compared to that of 1.974 hrs for the algorithm proposed in [105]. Similarly, when optimizing NDCG loss, our method leads to a total training time of 1.632 hrs, which is significantly better than the 2.217 hrs taken for training when using the method proposed in [14]. This indicates that using our method helps the layerwise training procedure scale much better.

3.6 Discussion

We provided a characterization of ranking based loss functions that are amenable to a quicksort based optimization algorithm for the loss augmented inference problem. We proved that our algorithm provides a better computational complexity than the state of the art methods for AP and NDCG loss functions and also established that the complexity of our algorithm cannot be improved upon asymptotically by any comparison based method. On the other hand, we demonstrated that it is still possible to reduce the constant factors of the complexity of AP loss by exploiting its special structure. We empirically demonstrated the efficacy of our approach on challenging real world vision problems. In future, we would like to explore extending our approach to other ranking based non-decomposable loss functions like those based on the F-measure or the mean reciprocal rank.

3.7 Appendix

3.7.1 Efficient Approximation of AP-SVM

In sections 3.3 and 3.4, we presented exact methods for efficient optimization of a class of rank-based loss functions that includes AP loss and demonstrated their efficacy in faster training of AP-SVMs. However, despite these improvements, AP-SVM might be slower to learn compared to simpler frameworks such as the binary SVM, which optimizes the surrogate 0-1 loss. The disadvantage of using the binary SVM is that, in general, the 0-1 loss is a poor approximation for the AP loss. However, the quality of the approximation is not uniformly poor for all samples, but depends heavily on their separability. Specifically, when the 0-1 loss of a set of samples is 0 (that is, they are linearly separable by a binary SVM), their AP loss is also 0. This observation inspires us to approximate the AP loss over the entire set of training samples using the AP loss over the subset of difficult samples. In this work, we define the subset of difficult samples as those that are incorrectly classified by a simple binary SVM.

Formally, given the complete input $\mathbf{X}$ and the ground-truth ranking matrix $\mathbf{R}^*$, we represent individual samples as $\mathbf{x}_i$ and their class as y_i. In other words, $y_i = 1$ if $i \in \mathcal{P}$ and $y_i = -1$ if $i \in \mathcal{N}$. In order to approximate the AP-SVM, we adopt a two stage strategy. In the first stage, we learn a binary SVM by minimizing the regularized convex upper bound on the 0-1 loss over the entire training set. Since the loss-augmented inference for 0-1 loss is very fast, the parameters $\mathbf{w}_0$ of the binary SVM can be estimated efficiently. We use the binary SVM to define the set of easy samples as $\mathbf{X}_e = \{\mathbf{x}_i | y_i \mathbf{w}_0^\top \phi(\mathbf{x}_i) \geq 1\}$. In other words, a positive sample is easy if it is assigned a score that is greater than 1 by the binary SVM. Similarly, a negative sample is easy if it is assigned a score that is less than -1 by the binary SVM. The remaining difficult samples are denoted by $\mathbf{X}_d = \mathbf{X} - \mathbf{X}_e$ and the corresponding ground-truth ranking matrix by $\mathbf{R}_d^*$. In the second stage, we approximate the AP loss over the entire set of samples $\mathbf{X}$ by the AP loss over the difficult samples $\mathbf{X}_d$ while ensuring that the samples $\mathbf{X}_e$ are correctly classified. In order to accomplish this, we solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\xi \\ \text{s.t.} \quad & \mathbf{w}^\top \Psi(\mathbf{X}_d, \mathbf{R}_d^*) - \mathbf{w}^\top \Psi(\mathbf{X}_d, \mathbf{R}_d) \geq \Delta(\mathbf{R}_d^*, \mathbf{R}_d) - \xi, \forall \mathbf{R}_d, \\ & y_i \left(\mathbf{w}^\top \phi(\mathbf{x}_i) \right) > 1, \forall \mathbf{x}_i \in \mathbf{X}_e. \end{aligned} \tag{3.15}$$

In practice, we can choose to retain only the top $k\%$ of $\mathbf{X}_e$ ranked in descending order of their score and push the remaining samples into the difficult set $\mathbf{X}_d$. This gives the AP-SVM more flexibility to update the parameters at the cost of some additional computation.

We demonstrate the effectiveness of our idea using the PASCAL VOC 2011 [30] action classification data set. We use structured SVM models as discriminant functions and use the standard poselet [63] activation features to define the sample feature for each person bounding box. The feature vector consists of 2400 action poselet activations and 4 object detection scores. We refer the reader to [63] for details regarding the feature vector. Table 3.3 shows the AP for the rankings obtained by binary SVM, AP-SVM and our approximate AP-SVM (AP-SVM-APPX) for 'test' set. The time required for computing the most violated ranking when averaged over the training for all the action classes was 0.2341 second for AP-SVM-APPX which is significantly less than the 0.566 second taken for AP-SVM. The same timing for binary-SVM was 0.1068 second. Therefore, it can be observed that the proposed approximate AP-SVM gives results that are comparable to that of AP-SVM while being significantly faster in training.

Object class	Binary SVM	AP-SVM	AP-SVM-APPX		
			k=25%	k=50%	k=75%
Jumping	52.580	55.230	54.660	55.640	54.570
Phoning	32.090	32.630	31.380	30.660	29.610
Playing instrument	35.210	41.180	40.510	38.650	37.260
Reading	27.410	26.600	27.100	25.530	24.980
Riding bike	72.240	81.060	80.660	79.950	78.660
Running	73.090	76.850	75.720	74.670	72.550
Taking photo	21.880	25.980	25.360	23.680	22.860
Using computer	30.620	32.050	32.460	32.810	32.840
Walking	54.400	57.090	57.380	57.430	55.790
Riding horse	79.820	83.290	83.650	83.560	82.390

Table 3.3: *Test* AP *for the different action classes of* PASCAL VOC *2011 action dataset. For* AP-SVM-APPX*, we report test results for 3 different values of k, which is the percentage of samples that are included in the easy set among all the samples that the binary* SVM *had classified with margin > 1.*

3.7.2 Empirical comparison of rank-based loss functions and 0-1 loss

For the action classification experiments on the PASCAL VOC 2011 data set, we report the performance of models trained by optimizing 0-1 loss as well as AP loss in Table 3.3. Specifically, we report the AP on the test set for each of the 10 action classes. Similarly, we also report the performance of models trained by optimizing 0-1 loss as well as NDCG loss, in terms of NDCG on the test set in Table 3.4.

For our object detection experiments, we report the detection AP in Table 3.5 for all the 20 object categories obtained by models trained using 0-1 loss as well as AP loss. For all object categories other than 'bottle', AP loss based training does better than that with 0-1 loss. For 15 of the 20 object categories, we get statistically significant improvement with AP loss trained models compared to those trained using 0-1 loss (using paired t-test with p-value less than 0.05). While optimizing AP loss for learning gives an overall improvement of 7.12% compared to when using 0-1 loss, for 5 classes it gives an improvement of more than 10%. The bottom 2 classes with the least improvement obtained by AP loss based training, 'chair' and 'bottle' seem to be difficult object categories to detect, with detectors registering very low detection APs. In conjunction with the overall superior performance of AP loss for learning model parameters, the efficient method proposed in this thesis makes a good case for optimizing AP loss rather than 0-1 loss for tasks like object detection.

Object class	0-1 loss	NDCG loss
Jumping	86.409	87.895
Phoning	73.134	76.733
Playing instrument	81.533	83.666
Reading	74.528	75.588
Riding bike	94.928	95.958
Running	93.766	93.776
Taking photo	74.058	76.701
Using computer	79.518	78.276
Walking	89.789	89.742
Riding horse	96.160	96.875

Table 3.4: *Performance of classification models trained by optimizing 0-1 loss and* NDCG *loss, in terms of* NDCG *on the test set for the different action classes of* PASCAL VOC *2011 action dataset. We conduct 5-fold cross-validation and report the mean* NDCG *over the five validation sets.*

Object category	0-1 loss	AP loss	Object category	0-1 loss	AP loss
Aeroplane	46.60	48.18	Dining-table	14.20	39.53
Bicycle	48.53	61.45	Dog	33.55	36.25
Bird	33.31	36.73	Horse	46.14	53.86
Boat	15.23	19.66	Motorbike	29.97	34.81
Bottle	6.10	1.01	Person	29.58	30.41
Bus	37.01	49.51	Potted-plant	21.27	23.03
Car	61.28	66.78	Sheep	11.65	32.20
Cat	38.12	40.77	Sofa	36.66	42.03
Chair	2.71	3.23	Train	29.71	37.10
Cow	21.06	38.52	TV-monitor	27.31	37.26

Table 3.5: *Performance of detection models trained by optimizing 0-1 loss and* AP *loss, in terms of* AP *on the test set for the different object categories of* PASCAL VOC *2007 test set.*

Chapter 4

Learning to round for discrete labeling problems

4.1 Introduction

A discrete labeling problem is defined over a set of random variables, each of which needs to be assigned a value from a discrete label set. An assignment of values to all the random variables is referred to as a labeling. The large number of putative labelings (exponential in the number of random variables) are quantitatively distinguished from each other by means of an energy function. The goal of the discrete labeling problem is to compute the labeling with the minimum energy.

The discrete labeling problem plays a key role in many areas of computer science. For example, in machine learning it is required for maximum *a posteriori* estimation in graphical models. In theoretical computer science, several classical tasks such as vertex cover and graph partitioning can be viewed as discrete labeling problems. While special cases of the problem can be solved exactly in polynomial time [92], in general it is known to be NP-hard. A popular approach to obtain an approximate solution is to formulate the discrete labeling problem as an integer program, which can then be relaxed to obtain an easy-to-solve continuous optimization problem. While the continuous relaxation is closely related to the original integer program, its optimal solution can be fractional (it often is). Thus, a key requirement for using continuous relaxations is the availability of an accurate *rounding procedure*, that is, a method that can convert the optimal fractional solution to a feasible integer solution with low energy value. The most popular family of rounding procedures is based on randomized algorithms, which generally pose rounding as sampling from a distribution parameterized by the fractional solution.

The design of sampling based rounding procedures generally involves highly sophisticated mathematical analysis to establish the expected value of the energy of the rounded solution in the worst-case. In special cases such as uniform metric labeling, a lot of simple procedures, backed by strong mathematical analysis, have been proposed. For example, Kleinberg and Tardos [55] proposed the interval rounding procedure, which was later generalized by Chekuri *et al.* [15] for truncated linear and quadratic labeling. While such an approach has important theoretical consequences (specifically, it establishes the computational complexity of various instances of discrete labeling problems), from a practitioners point of view it suffers from two major deficiencies. First, it is highly onerous and therefore cannot scale well as each new class of problems would require expert knowledge to design an appropriate rounding procedure. Second, it focuses on the worst-case scenario (for ease of analysis), which often does not occur in practice.

Contributions. In order to alleviate the aforementioned deficiencies, we propose a novel machine learning framework that learns to round the solutions of a continuous relaxation using a training data set. The key observation of our approach is that many randomized rounding procedures can be viewed as sampling from a joint distribution of two types of random variables: (i) variables whose marginal probability is provided by the optimal fractional solution of the relaxation; and (ii) a set of appropriately designed latent variables. Viewed in this way, the problem of rounding readily lends itself to machine learning techniques. In this work, we employ a deep neural network that consists of two stages. In the first stage, it projects a low-dimensional input (namely, the optimal fractional solution of a relaxation) to a potentially high-dimensional space of the aforementioned latent random variables. In the second stage, it projects the representation encoded by the latent random variables back to the original space of feasible fractional solutions, to give the output of the neural network. The integer solution of the problem is obtained by using the simplest rounding procedure on the output. The parameters of the deep neural network are estimated by optimizing a differentiable learning objective that minimizes the expected energy of the labeling for a given set of training samples. Our approach can be readily applied to a large class of existing relaxations, including those based on linear programming [15, 56], quadratic programming [80] and second-order cone programming [59, 71]. By learning the neural network for a given set of samples that implicitly define the data distribution, we obtain more suitable rounding procedures for real-world problems compared to the ones designed for the worst-case. We demonstrate the efficacy of our approach on several instances of the discrete labeling problem, including uniform metric labeling and truncated linear and quadratic labeling by comparing them to the hand-designed rounding procedures proposed in the theoretical computer science literature.

4.2 Related Work

As mentioned earlier, most of the literature on randomized rounding procedures focuses on hand-designed algorithms for special instances of the discrete labeling problem. Examples include interval rounding and hierarchical rounding for linear programming relaxations of metric labeling [15, 55], hyperplane rounding for semidefinite relaxations of graph partitioning [31], and contention resolution for multilinear relaxations for submodular maximization [99]. While such an approach is of great theoretical importance, the complexity of mathematical analysis and the focus on the worst-case makes it less appealing in practice.

Our work exploits the close relationship between randomized rounding and sampling. Similar to rounding, there have been several hand-designed sampling algorithms that have been proposed in machine learning [2, 10]. Furthermore, there has also been some effort in learning to sample using a training data set [83, 107]. However, to the best of our knowledge, ours is the first approach to exploit the connection between sampling and rounding for discrete labeling problems. Moreover, unlike sampling in which the main objective is to maximize sample fidelity with the original distribution, our goal is to minimize a task specific energy function by learning to round.

There is also a rich history in machine learning for learning latent spaces for a particular problem [6, 40, 42]. The one most closely related to our approach is the variational auto-encoder (VAE) [52], which uses a deep neural network in order to obtain an expressive latent representation of the input space. However, there are several key differences between our framework and the VAE. First, our training objective is designed to maximize the accuracy of rounding instead of minimizing the reconstruction error. Second, while the VAE aims to learn the parameters of a neural network that generalizes well across the entire data distribution, our problem is concerned with learning the parameters for each individual instance of the problem.

Finally, there has been work on using reinforcement learning for function optimization [5, 101]. Reinforcement learning also optimizes an entropy regularized expected energy. However, in our case, a key difference is that we compute the expected energy and its exact gradient analytically instead of relying on estimates of the policy gradient. This reduces the variance in the gradient, thereby enabling efficient learning. Another key difference is that, unlike the reinforcement learning based methods, we also condition our model with respect to a primal marginal solution which allows a trained model to generalize to unseen energy functions.

4.3 Preliminaries

Discrete Labeling Problem. A discrete labeling problem is defined over a set of random variables $\mathcal{X} = \{X_1, X_2, \cdots, X_n\}$, each of which can take a value from the label set $\mathcal{L} = \{l_1, l_2, \cdots, l_h\}$. An assignment of values to all the random variables is called a labeling, and is denoted by $\mathbf{x} \in \mathcal{L}^n$. The labelings are quantitatively distinguished from each other by the means of an energy function $E_\theta : \mathcal{L}^n \to \mathbb{R}$, which consists of a sum of potential functions. For the sake of clarity, we will restrict ourselves to a pairwise energy function. In other words, the energy of a labeling **x** is given by:

$$E_\theta(\mathbf{x}) = \left\{ \sum_{a \in [n]} \theta_a(x_a) + \sum_{(a,b) \in [n]^2} \theta_{ab}(x_a, x_b) \right\}. \tag{4.1}$$

Here, $[n] = \{1, ..., n\}$, and $\theta_a(x_a)$ and $\theta_{ab}(x_a, x_b)$ are short hand for $\theta_a(X_a = x_a)$ and $\theta_{ab}(X_a = x_a, X_b = x_b)$ respectively. While, $\theta_a(X_a = x_a)$ denotes the unary potential of assigning the label x_a to the random variable X_a, $\theta_{ab}(X_a = x_a, X_b = x_b)$ denotes the pairwise potential of assigning the labels x_a and x_b to random variables X_a and X_b respectively. The discrete labeling problem is specified as follows: $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x} \in \mathcal{L}^n} E_\theta(\mathbf{x})$. The general discrete labeling problem allows for higher-order potentials (that is, potentials that depend on the labels of an arbitrarily large subset of random variables). Our approach can be trivially generalized to handle such potentials by suitably modifying the training objective. We assume that the potentials are finite valued. In other words, all the labelings are valid, and the task is to identify the one with the minimum energy. This assumption is mainly due to the fact that we parameterize our framework via a neural network, which requires gradients for all possible parameters in order to utilize the back-propagation algorithm during training.

Integer Programming Formulation. The discrete labeling problem can be reformulated as an integer program using indicator variables $y_a(i) \in \{0, 1\}$ corresponding to all random variables $X_a \in \mathcal{X}$ and labels $l_i \in \mathcal{L}$. Each indicator variable $y_a(i) = 1$, if $X_a = l_i$ and 0 otherwise. Formally, the following program provides the minimum energy labeling:

$$\begin{aligned} \min \quad & \sum_{a \in [n]} \sum_{l_i \in \mathcal{L}} \theta_a(i) y_a(i) \\ & + \sum_{(a,b) \in [n]^2} \sum_{(l_i, l_j) \in \mathcal{L}^2} \theta_{ab}(i,j) y_a(i) y_b(j), \\ \text{s.t.} \quad & \sum_{l_i \in \mathcal{L}} y_a(i) = 1 \;\; \forall a \in [n], \\ & y_a(i) \in \{0, 1\} \;\; \forall a \in [n], \forall l_i \in \mathcal{L}. \end{aligned} \tag{4.2}$$

Here, the objective function is just a reformulation of the energy function in equation 4.1 using the binary indicator variables. The first constraint ensures that each random variable is assigned exactly one label, while the second constraint ensures that the optimization variables are binary.

Continuous Relaxations. The above integer program can be relaxed to obtain a continuous optimization problem. Several relaxations have been proposed in the literature, including linear programming [15, 56], quadratic programming [80] and second-order cone programming [59, 71]. All the aforementioned relaxations drop the integrality constraints, and instead enforce the variables $\mathbf{y}_a$ to belong to a probability simplex, that is, $y_a(i) \geq 0$ and $\sum_i y_a(i) = 1$. The resulting continuous problem is then solved to obtain an optimal fractional solution $\mathbf{y}^*$.

Randomized Rounding Procedures. Given a feasible fractional solution $\bar{\mathbf{y}}$, a rounding procedure treats $\bar{\mathbf{y}}_a(i)$ as the probability of assigning the label l_i to the random variable X_a. Several rounding procedures that satisfy this property have been proposed in the literature. We refer the interested reader to [97, 102] for examples. Although most often these algorithms are applied on optimal fractional solutions, the analysis applies to all feasible solutions. In our work, we will use the simplest rounding procedure to obtain integer solutions from the output of a deep neural network, which we refer to as complete rounding. The main steps of the complete rounding procedure are described in Algorithm 6. Complete rounding computes the cumulative distribution of $\mathbf{y}_a$ for each $X_a \in \mathcal{X}$ (step 3). It then samples from the cumulative distribution using the same real number $r \in [0, 1]$ for all the random variables (step 4). Note that the use of the same real number is important. Otherwise, the rounding procedure can be shown to produce arbitrarily bad labelings (see [58] for examples).

Algorithm 6 *The complete rounding algorithm.*

```
   Input: A fractional solution $\mathbf{y}$ of a relaxation.
1: Sample a real number $r$ from uniform distribution over $[0, 1]$.
2: for all $X_a \in \mathcal{X}$ do
3:     Define, $Y_a(0) = 0$, $Y_a(i) = \sum_{j=1}^{i} y_a(j)$.
4:     Assign label $l_i$ to random variable $X_a$ if     $Y_a(i-1) \leq r < Y_a(i)$.
```

4.4 Trainable Latent Variable Model For Rounding

Given a feasible (or optimal) fractional solution $\mathbf{y}$ of a continuous relaxation, randomized rounding procedures can be viewed as assigning a label to the set of discrete random variables $\mathcal{X}$ from the set $\mathcal{L}$ such that $\Pr(X_a = l_i) = y_a(i)$. There are several ways in which one can achieve this goal, including the simple complete rounding procedure described above. What separates a good rounding procedure from a bad one is the joint probability of the labeling of a subset of random variables. Since we have restricted our description to pairwise energy functions, the key criterion for differentiating two rounding procedures is the joint probability of assigning two random variables X_a and X_b to the labels l_i and l_j respectively.

To illustrate the weakness of complete rounding, consider the following simple example of a uniform metric labeling problem (also referred to as the Potts model) defined over n random variables, each of which can take 1 of n possible labels. The unary potential for the a^{th} random variable X_a is defined as follows:

$$\theta_a(i) = \begin{cases} \infty & \text{if } i = a, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Every pair of random variables X_a and X_b are assumed to be connected by an edge and the pairwise potential between them is defined as follows: $\theta_{ab}(i, j) = \delta(i \neq j)$. Any labeling that assigns the same label to all but one of the random variables would be optimal for this discrete labeling problem.

On the other hand, the optimal fractional solution $\mathbf{y}$ obtained for the LP relaxation of this discrete problem can be defined as follows:

$$y_a(i) = \begin{cases} 1/(n-1) & \text{if } i \neq a, \\ 0 & \text{otherwise.} \end{cases} , \forall a \in \{1, \ldots, n\} \tag{4.4}$$

It can be verified that, when performing complete rounding on this optimal fractional solution, if the random number r is between $(i-1)/(n-1)$ and $i/(n-1)$ for $i = 1, 2, \cdots, n-1$, the first i variables take the label $i+1$, and the remaining ones take the label i. Specifically, it is impossible for it to do so when the random number $r \in [1/(n-1), (n-2)/(n-1)]$. Therefore, the probability of the complete rounding procedure to output an optimal integral solution (that is assigning the same label to all but 1 of the random variables) is only $2/(n-1)$. In order to overcome the deficiency of complete rounding, Kleinberg and Tardos [55] proposed a more suitable rounding procedure for uniform metric labeling. In what follows, we provide a novel interpretation of their procedure based on latent variable models, which will motivate our general learning based framework.

Consider an augmented label set $\mathcal{L}' = \{l_0\} \cup \mathcal{L}$, where the auxiliary label l_0 indicates that a random variable has not yet been assigned a label. We define a latent variable Z, which can take a value from the label set $\mathcal{L}$. The probability $\Pr(Z = l_i) = 1/h$ for all $l_i \in \mathcal{L}$. Furthermore, we define $\Pr(X_a|Z)$ as

$$\Pr(X_a = l_i | Z = l_j) = \begin{cases} y_a(i) & \text{if } i = j \neq 0, \\ 1 - y_a(i) & \text{if } i = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

In order to obtain a labeling, we use an iterative procedure. At each iteration, we first sample from the distribution $\Pr(Z)$ to fix the value of the latent variable. Next, we use complete rounding on the distributions $\Pr(X_a|Z)$ for all random variables $X_a \in \mathcal{X}$. If a random variable is assigned a label $l_i \in \mathcal{L}$ (that is, not the label l_0), then we fix its label to l_i. For all the unassigned random variables (that is, those with the label l_0) we repeat the above process until a valid labeling has been obtained. The above iterative procedure can be viewed as sampling from the joint distribution $\Pr(Z, X_a) = \sum_{l_i \in \mathcal{L}} \Pr(Z = l_i) \Pr(X_a | Z = l_i)$, and marginalizing out the latent variable. It can be verified that, in the case of the illustrative example for uniform metric labeling, the above procedure always outputs an optimal integral solution. It can do so because of the use of latent variables. This can thus result in an improved expected energy of the output labeling compared to complete rounding.

At first sight, the choice of the latent variable Z, its distribution $\Pr(Z)$ and the conditional distributions $\Pr(X_a|Z)$ may appear arbitrary. However, there are two factors that governed their design. First, they have to exploit the structure of the pairwise potentials, which encourages a pair of random variables to be assigned the same label. Second, the latent variable and the corresponding distributions have to be simple enough to lend themselves to worst-case mathematical analysis. We now consider each of the two aforementioned factors to motivate our methodology. The first factor implies that, for every different family of the discrete labeling problems, we would need to design a new rounding procedure. Indeed, for truncated linear and quadratic labeling, the random variable represents an interval of consecutive labels, instead of a single label [15]. Given the vast number of choices, it is clear that the process of hand-designing a rounding procedure is too tedious and would not scale well to meet the demands of an increasingly automated world. However, our novel interpretation of rounding procedures as latent variable models opens the door to the use of powerful machine learning frameworks. The second factor implies that the simple form of distributions is not a requirement in practice as we are not interested in worst-case analysis. Thus, inspired by the recent success of deep latent variable models such as VAE [52], we propose to parameterize rounding procedures for discrete labeling problems using

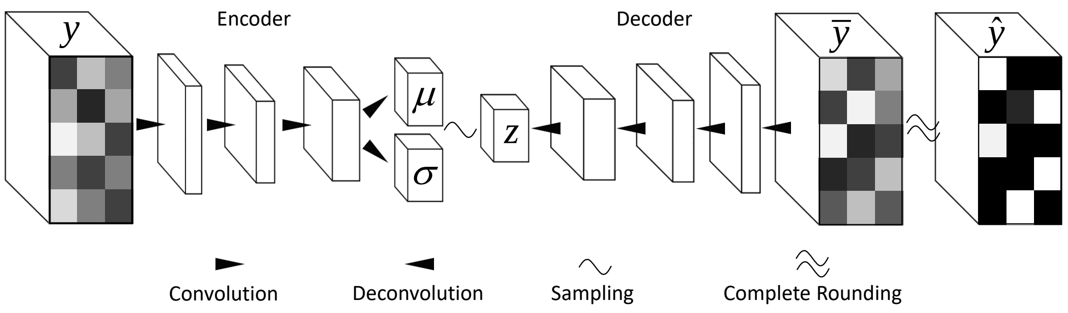


Figure 4.1: *Our deep latent variable model that includes an encoder, a latent layer and a decoder. The encoder takes in a feasible fractional solution* $\mathbf{y}$ *as input and computes the parameters* $[\boldsymbol{\mu}, \boldsymbol{\sigma}]$ *of the latent variable distribution* $\Pr(Z)$. *The decoder takes a sample from* $\Pr(Z)$ *and computes the output fractional solution* $\bar{\mathbf{y}}$. *Finally, approximate integral solution* $\hat{\mathbf{y}}$ *is obtained by performing complete rounding on* $\bar{\mathbf{y}}$.

a deep neural network. This allows us to learn highly complex latent variables and the corresponding distributions (which can depend on the fractional solution $\mathbf{y}$) using a set of training samples that implicitly define the data distribution of interest. In what follows, we describe our model and its end-to-end differentiable training objective in detail.

4.4.1 Prediction Using Deep Latent Variable Model

We use deep neural networks to model both the distribution $\Pr(Z)$ associated with the latent random variables Z and the conditional distribution $\Pr(X|Z)$ that projects back into the space of feasible solutions. Overall, our network is composed of an encoder ($\mathcal{E}_{\boldsymbol{\alpha}}$), a layer of latent variables (Z) and a decoder ($\mathcal{D}_{\boldsymbol{\beta}}$). Here, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the learnable parameters of the encoder and decoder respectively. Figure4.1 shows a representation of our deep latent variable model. While we have shown a fully convolutional network here, any network architecture can be used for this purpose.

The detailed description for rounding using our deep latent variable model is outlined in Algorithm 7. First, the parameters $\boldsymbol{\phi}$ of the latent variable distribution $\Pr(Z;\boldsymbol{\phi})$ are computed as outputs by the encoder $\mathcal{E}_{\boldsymbol{\alpha}}$, that is, $\boldsymbol{\phi}(\mathbf{y}) = \mathcal{E}_{\alpha}(\mathbf{y})$ (step 1). We then sample $\mathbf{z}$ from the distribution $\Pr(Z;\boldsymbol{\phi})$ (step 2) and pass it through the decoder $\mathcal{D}_{\boldsymbol{\beta}}$ to get an output fractional solution $\bar{\mathbf{y}} = \mathcal{D}_{\boldsymbol{\beta}}(\mathbf{z})$ (step 3). This output fractional solution $\bar{\mathbf{y}}$ parameterizes the distribution $\Pr(X = \hat{\mathbf{y}}|Z;\bar{\mathbf{y}})$, where $\hat{\mathbf{y}}$ is a feasible integral solution. We perform complete rounding as described in Algorithm 6 on $\bar{\mathbf{y}}$ to get our output $\hat{\mathbf{y}}$

(step 4). In practice, we perform several iterations of complete rounding on $\bar{\mathbf{y}}$ and choose $\hat{\mathbf{y}}$ to be the integral solution with the lowest energy $E_\theta(\hat{\mathbf{y}})$. In our experiments, we parameterize the latent variable distribution $\Pr(Z; \boldsymbol{\phi})$ as a diagonal Gaussian with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$, that is, $\boldsymbol{\phi} = [\boldsymbol{\mu}, \boldsymbol{\sigma}]$. However, one can also assume other parameterized models like the Bernoulli distribution for the latent variables.

4.4.2 Training Methodology

Our aim here is to learn the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the encoder and the decoder respectively using training set which is assumed to contain independently sampled instances from the data distribution. The training data set D can consist of several instances of any particular discrete labeling problem that vary in terms of the values for the parameters of the energy function E_θ, while maintaining the same underlying structure. For example, in case of the problem of semantic segmentation of images, we can have different images as the samples of the training set while assuming the same structure for inter-pixel dependencies for all the images. To be specific, each instance s_t in the training set $D = \{s_1, \ldots, s_N\}$ should include the parameters θ_t of the energy function and a feasible fractional solution $\mathbf{y}_t$ from a continuous relaxation of the original discrete labeling problem.

Algorithm 7 *Rounding using deep latent variable model.*

```
    Input: A feasible solution $\mathbf{y}$ of the continuous relaxation.
1:  Compute parameters of the latent variable distribution by passing through the encoder:
        $\boldsymbol{\phi}(\mathbf{y}) = \mathcal{E}_\alpha(\mathbf{y})$.
2:  Sample $\mathbf{z}$ from the latent variable distribution $P(Z; \boldsymbol{\phi})$.
3:  Compute output fractional solution by passing $\mathbf{z}$ through the decoder: $\bar{\mathbf{y}} = \mathcal{D}_{\boldsymbol{\beta}}(\mathbf{z})$.
4:  Perform complete rounding as described in Algorithm 6 to obtain an integral solution $\hat{\mathbf{y}}$ corresponding to $\bar{\mathbf{y}}$.
    Output: An approximate integral solution $\hat{\mathbf{y}}$ to the original discrete labeling problem.
```

As has been discussed elaborately in the previous sections, our objective is to find the integral solution $\hat{\mathbf{y}}$ with the lowest possible energy $E_\theta(\hat{\mathbf{y}})$. Therefore, given the training data, we should try to minimize the energy $E_\theta(\hat{\mathbf{y}})$ of the output integral solutions for all of the training examples. However, since ours is a stochastic model, it is appropriate for us to minimize the expected energy with respect to the distribution $\Pr(X; y_t, \boldsymbol{\alpha}, \boldsymbol{\beta})$ for all training samples $s_t \in \mathcal{D}$.

Given a training sample $s_t \in \mathcal{D}$ and a value $\mathbf{z}$ from the latent variable distribution $\Pr(Z; \boldsymbol{\phi}(y_t, \boldsymbol{\alpha}))$, we can sample several integral solutions using the complete rounding procedure. The expectation with respect to the distribution induced by the complete rounding procedure can be evaluated exactly by using the expression presented in *Lemma 11* of [58]. To be specific, the expected pairwise energy can be computed as,

$$\mathbb{E}(\theta_{ab}(\hat{\mathbf{y}}_a, \hat{\mathbf{y}}_b)) = \sum_{i=1}^{h-1} Y_a(i)\Theta_1(i) \tag{4.6}$$
$$+ \sum_{j=1}^{h-1} Y_b(j)\Theta_1(j) + \sum_{i=1}^{h-1}\sum_{j=1}^{h-1} |Y_a(i) - Y_b(j)|\Theta_2(i,j).$$

Here, Y_a and Y_b are the cumulative distributions with respect to $\mathbf{y}_a$ and $\mathbf{y}_b$, and Θ_1 and Θ_2 are defined as follows,

$$\Theta_1(i) = \tfrac{1}{2}(\theta_{ab}(l_i, l_1) + \theta_{ab}(l_i, l_h) - \theta_{ab}(l_{i+1}, l_1)$$
$$- \theta_{ab}(l_{i+1}, l_h)), \forall i \in \{1, .., h-1\}, \tag{4.7}$$
$$\Theta_2(i,j) = \tfrac{1}{2}(\theta_{ab}(l_i, l_{j+1}) + \theta_{ab}(l_{i+1}, l_j) - \theta_{ab}(l_i, l_j)$$
$$- \theta_{ab}(l_{i+1}, l_{j+1})), \forall i, j \in \{1, .., h-1\}. \tag{4.8}$$

The overall expected energy of the output integral solution $\hat{\mathbf{y}}$ with respect to complete rounding then can be computed as,

$$\mathop{\mathbb{E}}_{\hat{\mathbf{y}} \sim \Pr(X|Z;\boldsymbol{\beta})} [E_\theta(\hat{\mathbf{y}})] = \textstyle\sum_{X_a \in \mathcal{X}_a} \langle \theta_a(\hat{\mathbf{y}}_a), \mathbf{y}_a \rangle \tag{4.9}$$
$$+ \textstyle\sum_{(X_a, X_b) \in \mathcal{X}^2} \mathbb{E}(\theta_{ab}(\hat{\mathbf{y}}_a, \hat{\mathbf{y}}_b)).$$

As discussed above, the expected energy of the output integral solution $\hat{\mathbf{y}}$, with respect to the distribution induced by the complete rounding procedure, has a closed form expression. This allows us to analytically compute the gradient of this expected energy with respect to the output fractional solution $\bar{\mathbf{y}}$ and thus have no variance in gradient estimation.

For our training objective, we further take expectation of the above computed expected energy with respect to the latent variable distribution $\Pr(Z; \boldsymbol{\phi})$ to obtain,

$$\mathop{\mathbb{E}}_{\hat{\mathbf{y}} \sim \Pr(X; y_t, \boldsymbol{\alpha}, \boldsymbol{\beta})} [E_{\theta_t}(\hat{\mathbf{y}})]$$
$$= \mathop{\mathbb{E}}_{\mathbf{z} \sim \Pr(Z; \boldsymbol{\phi}(y_t, \boldsymbol{\alpha}))} \left[\mathop{\mathbb{E}}_{\hat{\mathbf{y}} \sim \Pr(X|Z=\mathbf{z};\boldsymbol{\beta})} [E_{\theta_t}(\hat{\mathbf{y}})] \right]. \tag{4.10}$$

Since analytical computation of this expected energy is not feasible, we have to estimate it by using k samples from the latent variable distribution. When the latent variable distribution is chosen to be a Gaussian distribution, the gradient of this sampling based estimation of the expected energy with respect to the parameters of the distribution can be computed using the reparameterization trick. This results in very low variance for the gradient estimation. In the general case, the reinforce algorithm can be used to estimate the gradient.

Finally, our training objective is obtained by taking expectation of the above computed expected energy with respect to the data distribution $\mathcal{D}$. As in case of all machine learning frameworks, we estimate the expectation over the data distribution by using samples from the training data set D. Then the training objective function can be written as

$$J_E(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathop{\mathbb{E}}_{s_t \sim D} \left[\mathop{\mathbb{E}}_{\hat{\mathbf{y}} \sim \Pr(X; y_t, \boldsymbol{\alpha}, \boldsymbol{\beta})} [E_{\theta_t}(\hat{\mathbf{y}})] \right] \tag{4.11}$$

While optimizing the expected energy, the model might sometime get stuck in local minima corresponding to integral $\bar{\mathbf{y}}$'s that lead to bad approximate solutions. In order to avoid such bad local minima during optimization, we regularize the output fractional solutions $\bar{\mathbf{y}}$ to bias them towards having higher entropy. Specifically, we add the following entropy based regularization term to our objective function,

$$\begin{aligned} & J_R(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \\ & \mathop{\mathbb{E}}_{s_t \sim D} \left[\mathop{\mathbb{E}}_{\mathbf{z} \sim \Pr(Z; \boldsymbol{\phi}(y_t, \boldsymbol{\alpha}))} \left[\sum_{a \in [n]} \bar{\mathbf{y}}_a(\mathbf{z}, \boldsymbol{\beta}) \log \bar{\mathbf{y}}_a(\mathbf{z}, \boldsymbol{\beta}) \right] \right] . \end{aligned} \tag{4.12}$$

Overall, we optimize the objective function $J = J_E + \lambda J_R$ for learning the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the encoder and the decoder respectively. Here, λ is a hyper-parameter which we fix appropriately. Since, this objective function is differentiable, we can use the standard back-propagation algorithm to train our network parameters. In order to back-propagate through the sampling process at the latent layer, we use the re-parameterization trick for Gaussian distribution as proposed in [52].

4.5 Experiments

We demonstrate the efficacy of our approach, described in the previous section, on discrete labeling problems using both synthetic as well as real world data.

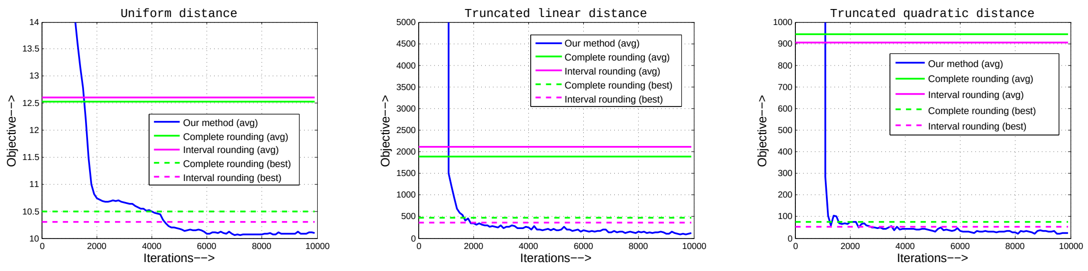


Figure 4.2: *Objective function Vs. iterations during optimization over the training set of the synthetic data set. Here, 'avg' refers to expected energy (E_{avg}) and 'best' refers to minimum energy (E_{min}).*

Labeling Metric	Max Rounding	Complete Rounding		Interval Rounding		Our method	
	E_{avg}=E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}
Uniform	12.204	12.532	10.435	12.437	10.327	10.180	10.103
Truncated Linear	74.423	1887.213	19.427	1936.361	17.787	153.801	14.762
Truncated Quadratic	152.532	904.821	84.640	876.610	72.418	71.469	65.359

Table 4.1: *Average (E_{avg}) and minimum (E_{min}) energy obtained by the different methods on the test set for the discrete labeling problem on synthetic data.*

4.5.1 Discrete Labeling Of Densely Connected Graphs

Problem. We consider the problem of labeling a set of 25 random variables from a discrete set of 21 labels. We assume that all the random variables are dependent on each other and these dependencies are encoded by a densely connected graph. We consider three different types of distance metrics for encoding label compatibility, namely, the uniform, truncated linear and truncated quadratic distances. The uniform distance metric is also known as the Potts model and is defined as $d(i, j) = \delta(i \neq j)$. Where as, the truncated linear and truncated quadratic distances are defined as $d(i, j) = \min(|i-j|, M)$ and $d(i, j) = \min(|i - j|^2, M)$ respectively. We use $M = 10$ for our experiments. We also assume a spatial arrangement of the random variables in a 5×5 lattice such that each vertex v_a has a location coordinate $c_a = (x_a, y_a)$. As such, the pairwise potential associated with the edge between vertices v_a and v_b is defined as $\theta_{ab}(i, j) = w_{ab}d(i, j)$, where, $w_{ab} = \exp(-||c_a - c_b||_2^2/\sigma_s^2)$ is a spatial Gaussian weight with scaling factor σ_s.

Dataset. We use a synthetically generated collection of 100 graphs for this experiment. For each graph, we randomly generate the unary potential associated with each vertex from a uniform distribution

Max Rounding	Complete Rounding		Interval Rounding		Our method		Our method (with finetuning)	
E_{avg}=E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}
803.417	742.596	696.452	744.249	691.179	658.936	653.255	655.547	652.836

Table 4.2: *Average (E_{avg}) and minimum (E_{min}) energy obtained by the different methods on the test set for the task of semantic segmentation on MSRC data set.*

Max Rounding	Complete Rounding		Interval Rounding		Our method		Our method (with finetuning)	
E_{avg}=E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}	E_{avg}	E_{min}
977.203	1131.940	977.242	1077.385	968.124	949.794	936.316	945.433	935.725

Table 4.3: *Average (E_{avg}) and minimum (E_{min}) energy obtained by the different methods on the test set for the task of semantic segmentation on MSRC data set when the input fractional solution corresponds to the negative exponentiated min-marginal provided by TRW-S.*

over $[0, 1]$. We also set the parameters σ_s and a weighing factor τ for the pairwise potentials, from a uniform distribution over $[-10^5, 10^5]$. From our collection of 100 graphs, we use 50 for training and 50 for testing.

Methods. We train a deep latent variable model as described in Section 4.4.1 with both the encoder and the decoder being modeled by neural networks composed of fully connected layers. We use the efficient proximal LP solver proposed in [1] to obtain fractional solutions for training our model. The scaling factor for the entropy regularization term in our training objective function is fixed to $\lambda = 1$ for all our experiments. We compare our method with other standard rounding procedures like max-rounding, complete rounding and interval rounding[15, 55].

Results. Figure 4.2 shows the progression of the objective function when our model is being optimized on the training data set. As can be seen, our model swiftly learns to do accurate rounding and outperforms the hand designed rounding procedures like complete and interval rounding. Even though this performance corresponds to the training set, it is significant because unlike training for tasks like classification, we don't use any kind of ground-truth information. However, it is not necessary to train the model from scratch for each new sample. Instead, we only finetune the trained model for the new sample by optimizing over it for a very few iterations (20 in this case). Such finetuning is possible because we do not need any kind of ground-truth information for it. To evaluate, for each test sample, we use different randomized rounding methods, including ours, to sample 1000 integral solutions and compute the expected energy E_{avg} and the minimum energy E_{min} over this set. We report the mean values of E_{avg} and E_{min} computed over the entire test set. Table 4.1 shows the results for the uniform, truncated linear and truncated quadratic labeling distance metrics. As can be seen, our method outperforms the other rounding procedures in all the 3 cases.

4.5.2 Semantic Segmentation of Images

Problem. We consider the problem of semantic segmentation of images which can be formulated as a labeling task in which each pixel has to be assigned a label from the set of semantic classes. We formulate this problem as a labeling problem over a grid graph in which each vertex v_a corresponds to a particular pixel and is connected to its four immediate neighbors via edges. The pairwise potential associated with an edge between vertices v_a and v_b is defined as $\theta_{ab}(i, j) = w_{ab}\delta(i \neq j)$. Here, $w_{ab} = \exp(-||f_a - f_b||_2^2/\sigma_c^2)$ is a color based Gaussian weight with f_a being the 3D color vector of the pixel a and σ_c being a scaling factor.

Dataset. We use the MSRC data set [84] for this experiment. It consists of a total of 591 images with ground-truth segmentation. We use the standard data set split of 276 training, 59 validation and 256 test samples, as specified in [84]. We use the texton based features proposed in [84] for unary potentials in our experiments.

Methods. For this task, the encoder and the decoder for our method are modeled as fully convolutional networks, in order to allow for variable input image sizes. The hyper-parameters σ_c and a weighing factor τ for the pairwise potentials is set by using the validation set. Here, we apply rounding to two different sets of fractional solutions: 1) Primal solutions obtained by solving a continuous relaxation [80] of the original primal problem, 2) Primal feasible solutions computed using expected min-marginals obtained from the TRW-S algorithm. Specifically, for each data set sample, we use fractional solutions obtained using the efficient QP solver proposed in [23] and those corresponding to the negative exponentiated min-marginal provided by TRW-S. We compare rounding procedures like max-rounding, complete rounding and interval rounding [15, 55].

Results. Similar to the previous experiment, we report the expected energy E_{avg} and the minimum energy E_{min} on the test set and compare our method with complete and interval rounding. For results shown in the second last column of Table 4.2 and Table 4.3, we do not fine-tune our model for the test samples and report results by simply forward passing through the network learned with the training set. However, our model is still able to generalize well to the test samples and outperforms the other rounding procedures for both types of fractional solutions. We also report results for the case when we fine-tune our model for each of the test samples. As can be seen from Table 4.2 and Table 4.3, fine-tuning slightly improves the results in both the cases.

4.6 Discussion

We proposed a novel framework for performing rounding for discrete labeling problems. Our approach views rounding as a method of sampling from a latent variable model and employs deep neural networks for this purpose. We showed that our method can adapt to different problem structures and outperforms hand designed rounding procedures on these tasks. Going ahead, we would like to explore approaches for performing rounding in presence of constraints.

Chapter 5

Conclusion and future work

In this thesis we investigated into some problems at the intersection between optimization and machine learning. In the following sections we discuss our contributions in this space and some interesting problems that can be explored in future.

5.1 Contributions of the thesis

The learning problem in a machine learning framework can be formulated as an optimization problem, the difficulty of which critically depends on the complexity of the variable whose value we are interesting in predicting, and the complexity of the criteria that is used for learning the model. When the output variable is a complex object, it might be possible to represent the possible values it can take using a very high dimensional continuous space or a very large structure-less discrete set. However, doing so would require us to search through an exponentially large search space to solve the learning optimization problem. So, it becomes critical to design representations for the output space and corresponding optimization algorithms that can leverage the structure inherent to the task at hand and solve the learning optimization problem efficiently. In a similar vein, using generic solvers to optimize complex loss functions can be highly inefficient and sometimes even infeasible. In this context, designing efficient optimization algorithms that can leverage the structure of specific types of loss functions is an important problem.

In **chapter 2**, we note that within the margin-maximizing framework of SVM, problems of progressively complicated output spaces can be formulated in terms of binary-SVM, multi-class-SVM and structured-SVM. We proposed a novel partial-linearization based approach for optimizing the multi-class and structured SVM learning problems [66]. Our method was an intuitive generalization of the Frank-Wolfe [44] and the exponentiated gradient [16] algorithms. In particular, it allowed us to combine several of their desirable qualities like an expectation oracle, optimal step-size and a block coordinate formulation, into one approach. This partial linearization based method allows one to solve large-scale problems and can be potentially adapted to also train models based on neural networks using a margin maximizing criterion similar to that of an SVM.

While complex output spaces certainly increase the complexity of the learning optimization problem, the structure of the loss function that is used as a criterion for learning model parameters is also crucial. This is particularly true for cases where the output is a structured object like a ranking and it is critical to choose a sophisticated enough loss function that can effectively differentiate between subtly different outputs. However, popular loss functions like AP loss and NDCG loss that have been extensively used as evaluation measures for evaluating information retrieval systems, have not been as popularly used for learning the model parameters of these systems. This is because of the difficulty in optimizing these non-decomposable loss functions which is generally circumvented by either optimizing a structured hinge-loss upper bound to the loss function [105] or by using asymptotic methods like the direct-loss minimization framework [86]. Yet, the high computational complexity of loss-augmented inference, which is necessary for both the frameworks, prohibits its use on large training data sets.

In **chapter 3**, we made a key observation that while popular non-decomposable loss functions can not be additively decomposed onto individual samples, a wide class of such loss functions can be instead additively decomposed onto the negative samples. Further, many of those rank-based loss functions do not depend on the relative order of positive or negative samples among themselves. Rather, the loss for a ranking, depends only on the interleaving rank of positive and negative samples corresponding to the ranking. We showed that these key properties in a loss function allows for an efficient quicksort flavored divide and conquer algorithm to solve the loss augmented problem. We formally define the class of loss functions that allow for such a quicksort flavored algorithm as QS-*suitable* loss functions. The fact that popular rank-based loss functions like AP and NDCG loss belong to this class establishes the usefulness for such a characterization.

We developed a novel quicksort flavored algorithm for solving the loss-augmented inference problem efficiently for this class of loss functions [69]. We showed that our approach has a superior runtime of $O(|\mathcal{N}| \log |\mathcal{P}| + |\mathcal{P}| \log |\mathcal{N}|)$ compared to $O(|\mathcal{P}|\,|\mathcal{N}| + |\mathcal{N}| \log |\mathcal{N}|)$ of [14, 105], where, $\mathcal{P}$ and $\mathcal{N}$

denote the sets of positive and negative samples respectively. Moreover, we also establish that any comparison based algorithm would require $\Omega(|\mathcal{P}| \log |\mathcal{N}| + |\mathcal{N}| \log |\mathcal{P}|)$ operations. While it provides an asymptotic lower bound for comparison based algorithms, it does not rule out the possibility of improving the constants hidden within the asymptotic notation for a given loss function. We actually exploit the additional structure of the AP loss to further speed-up our algorithm [68]. Rather surprisingly, we show that in case of some models, parameter learning by optimizing complex non-decomposable AP and NDCG loss functions can be carried out faster than by optimizing simple decomposable 0-1 loss. Specifically, while each loss-augmented inference call is more expensive for AP and NDCG loss functions, it can take fewer calls in practice to estimate the parameters of the corresponding model.

In **chapter 4**, we explored the interesting reverse problem of learning optimization algorithms. In this space we primarily focused on using machine leaning techniques for improving combinatorial optimization algorithms. Specifically, we considered the framework of *relaxation+rounding* for solving discrete labeling problems. In such a framework, traditionally sampling based rounding methods have been designed by human experts. However, this demands extensive expert knowledge of the specific task and while the designed procedures might come with certain worst case theoretical guarantees, they might not be that practically effective. In order to alleviate the aforementioned deficiencies, we proposed a novel machine learning framework that learns to round the solutions of a continuous relaxation using a training data set [67]. The key observation of our approach was that many randomized rounding procedures can be viewed as sampling from a latent variable model. Viewed in this way, the problem of rounding readily lends itself to machine learning techniques. Our approach can be readily applied to a large class of existing relaxations, including those based on linear programming [15, 56], quadratic programming [80] and second-order cone programming [59, 71].

5.2 Future work

5.2.1 Partial linearization based optimization

The partial linearization based algorithm we proposed in chapter 2 is for learning of Support Vector Machines (SVM). It would be interesting to explore the plausible application of the partial linearization based method towards learning of deep neural networks. Learning of deep neural networks is a complex problem and traditionally the community has mainly relied on first order methods like stochastic gradient descent (SGD). However, the performance of such gradient based methods depends on clever ways to modulate the learning-rate across iterations. Because of the highly non-convex nature of the optimization problem associated with the learning of deep neural networks, it is challenging to come up with a hyperparameter-free learning-rate schedule. There have been some good work in this direction with a plethora of adaptive gradient based methods being proposed [24, 53, 106]. While they have become the go-to methods for learning deep neural networks, it has been shown that such methods obtain worse generalization than SGD [103]. Works like Deep Frank-Wolfe [9] try to bridge this gap between SGD and the adaptive gradient methods by leveraging the structure of the problem and breaking down the problem into linear-SVM sub-problems. In this context, considering that the partial linearization algorithm proposed in chapter 2 is a generalization of the Frank-Wolfe algorithm, it would be interesting to explore the application of the partial-linearization based optimization to train deep neural networks.

5.2.2 Efficient optimization of rank-based loss functions

In chapter 3, we presented an efficient quicksort flavored algorithm for a large class of non-decomposable rank-based loss functions. We also provided the complete characterization of the loss functions that are amenable to the proposed algorithm. This class of loss functions that include AP loss and NDCG loss have the crucial property of being decomposable onto negative samples which our algorithm takes advantage of for efficient optimization. Going ahead, it would be interesting to explore similar useful structures for other loss functions like F1-score and mean reciprocal rank that can be leveraged for optimization. For example it might be possible to decompose loss functions in other useful ways where each term depends on multiple samples but still has a structure that can practically aid in optimization.

In our work we focus on the loss augmented inference (LAI) problem associated with the different non-decomposable rank-based loss functions. This makes sense as the LAI problem is the main computational bottleneck in optimizing the popular structured hinge-loss upperbound to these loss functions. We optimize a surrogate to the loss function like the hinge-loss upperbound because the original loss functions like AP loss and NDCG loss are piecewise constant and lack any useful local gradient information that can be used for their direct optimization. While traditionally these surrogates have been expert designed, it would be of interest to explore learning techniques to aid in constructing differentiable surrogates to non-differentiable non-decomposable loss functions. There are some work in this direction, for example, [28] tries to learn a differentiable substitute for the sorting procedure and as a result construct differentiable surrogate of several rank-based loss functions. It would be of interest to further explore this direction of work aided by recent developments in the area of learning-to-optimize.

5.2.3 Learning to optimize for discrete labeling problems

In chapter 4, we presented a method for learning a stochastic rounding procedure by modelling it as sampling from a learnable latent variable model. Such an approach allows for adapting the rounding procedure to new domains without having to rely on expert effort to design domain specific procedures. While in this work we focused on the rounding module of the overall relaxation+rounding optimization pipeline, a good direction for future research would be to try and marry different ideas developed in context of relaxation methods and rounding procedures with the aim of improving the overall performance. It would also be interesting to explore approaches for performing rounding in presence of constraints.

5.2.4 Further explorations into optimization for and by machine learning

Traditionally machine learning as a field of research has always maintained a keen focus on leveraging the structure of the problem at hand to design models that are accurate and efficient. In this context, leveraging the structure involved going beyond the black box understanding of the problem and incorporating expert knowledge of the domain into the model itself. For example, if we consider the case of semantic segmentation of images, the output consists of simultaneous class label predictions for all pixels in the input image. Intuitively, the class label that a particular pixel is assigned is dependent on the label assigned to other pixels in its neighbourhood. As such, there is a dependency structure among the output variables. Over the years, many frameworks have been developed that try to encode this dependency structure in the output space [32, 93]. A typical framework would include a module that makes independent predictions for each individual variable and another module, most often in the form of a probabilistic graphical model, that enforces the expert defined structure to make the final prediction. Such models have traditionally been very successful by effectively combining knowledge learned from data with prior domain knowledge.

In recent years, deep neural networks have taken a slightly different route to success. They have significantly pushed the state-of-the-art in many machine learning tasks including those involving structured output spaces like semantic segmentation and machine translation [26, 104] primarily powered by the availability of huge amounts of data and computational resources. However, most deep models that have been popularly used for tasks like semantic segmentation or machine translation, do not explicitly try to leverage the structure of the output space to guide the search for the optimal solution. Instead, they make independent predictions for each output variable and rely on the representational power of a deep neural net to implicitly learn the structure of the output space from the data. Such an approach demands access to huge amounts of data and computational resources. While this might be possible in certain settings and tasks, there is an obvious case for more explicit graph based encoding of the domain knowledge into the model, so that it does not have to be learned from data. There are two primary reasons that discourages the use of graphical model based encoding of domain knowledge in deep learning frameworks: (i) Limited flexibility in the types of prior knowledge that can be effectively encoded using graphs, (ii) Difficulty in end-to-end learning with pipelines that include graph inference algorithms. Both these problems lie at the intersection of machine learning and optimization, and present interesting directions for research.

The first problem discourages the use of graphical models for encoding prior knowledge because with limited options for the types of distributions that the graphical model can be used to represent, it becomes very difficult to accurately encode the prior knowledge. An inaccurate encoding of the prior then generally hampers the performance of the overall pipeline. The class of distributions that a graphical model can represent is generally restricted by the efficiency of the associated inference algorithms. For example, it is desirable to have tree structured graphs and sub-modular dependency structures because it is easy to do inference on such graphs. While such graphs might be expressive enough for certain tasks, they generally fall short in accurately modeling the prior knowledge for most real world problems. As such, if efficient inference algorithms can be designed for more complicated graphs, then it would drastically increase the scope of applicability of graphical models. Since most graph inference problems can be formulated as combinatorial optimization algorithms, it seems like a promising problem for which learning techniques can be effectively used. It would be an interesting problem to parameterize the space of inference algorithms, possibly as Markov decision processes and then try to learn the model parameters, either by using some form of supervised learning or reinforcement learning.

While having efficient inference algorithms for a graphical model allows for easy predictions and can be used as a post processing step in deep learning pipelines, it still does not necessarily allow for end-to-end learning. This is because most of the optimization algorithms that are used for learning the model parameters of a deep neural network, rely on being able to compute the gradient of the task specific loss function with respect to the network parameters. This requires the different computational steps in a deep learning pipeline to be piece-wise differentiable and restricts the types of computational modules that can be included in the pipeline. Graph inference algorithms, which can often be formulated as combinatorial optimization procedures, are a category of computational modules that can make learning intractable in deep models. It is important therefore to develop methods for estimating the gradient of graph inference algorithms. In this context, the fact that graph inference procedures can be formulated as combinatorial optimization algorithms, actually indicates a direction which seems promising. It would be interesting to try and use the blackbox back-propagation framework that was recently developed for combinatorial optimization algorithms [98]. It would help us to construct an interpolation on the piecewise constant function that is characteristic of graph inference procedures and estimate an effective gradient that can be back-propagated. Such an approach would allow back-propagation through the graph inference algorithms and can make end-to-end learning possible in a deep learning framework with graphical models.

Bibliography

[1] T. Ajanthan, A. Desmaison, R. Bunel, M. Salzmann, P. Torr, and M P. Kumar. Efficient linear programming for dense CRFs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[2] C. Andrieu, N. De Freitas, A. Doucet, and M. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 2003.

[3] B. Bartell, G. Cottrell, and R. Belew. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[4] A. Behl, C. V. Jawahar, and M. P. Kumar. Optimizing average precision using weakly supervised data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[5] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.

[6] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[7] K. P. Bennett and E. Parrado-Hernández. The interplay of optimization and machine learning research. *The Journal of Machine Learning Research*, 2006.

[8] L. Berrada, A. Zisserman, and M. P. Kumar. Trusting SVM for piecewise linear CNNs. In *Proceedings of the International Conference on Learning Representations*, 2017.

[9] L. Berrada, A. Zisserman, and M. P. Kumar. Deep frank-wolfe for neural network optimization. In *Proceedings of the International Conference on Learning Representations*, 2019.

[10] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[11] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[12] C. Burges, R. Ragno, and Quoc V Le. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, 2007.

[13] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the International Conference on Machine Learning*, 2004.

[14] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2008.

[15] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[16] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *The Journal of Machine Learning Research*, 2008.

[17] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 1995.

[18] A. Cotter, H. Jiang, M. Gupta, S. Wang, T. Narayan, S. You, and K. Sridharan. Optimization with non-differentiable constraints with applications to fairness, recall, churn, and other goals. *Journal of Machine Learning Research*, 2019.

[19] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2002.

[20] G. B. Dantzig. Linear programming and extensions. *Princeton University Press*, 1963.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[23] A. Desmaison, R. Bunel, P. Kohli, P. Torr, and M. P. Kumar. Efficient continuous relaxations for dense CRFs. In *Proceedings of the European Conference on Computer Vision*, 2016.

[24] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of machine learning research*, 2011.

[25] E. Eban, M. Schain, A. Mackey, A. Gordon, R. Rifkin, and G. Elidan. Scalable learning of non-decomposable objectives. In *Proceedings of the International Conference on Artificial intelligence and statistics*, 2017.

[26] S. Edunov, M. Ott, M. Auli, and D. Grangier. Understanding back-translation at scale. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2020.

[27] J. Engel. Polytomous logistic regression. *Statistica Neerlandica*, 1988.

[28] M. Engilberge, L. Chevallier, P. Pérez, and M. Cord. Sodeep: a sorting deep net to learn ranking loss surrogates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[29] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[30] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *The International Journal of Computer Vision*, 2010.

[31] U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for max cut. *Random Structures & Algorithms*, 2002.

[32] P.F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[33] S. Fothergill, H. Mentis, P. Kohli, and S. Nowozin. Instructing people for training gestural interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[34] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 1956.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[36] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[37] G. Goh, A. Cotter, M. Gupta, and M. Friedlander. Satisfying real-world goals with dataset constraints. In *Advances in Neural Information Processing Systems*, 2016.

[38] T. Hazan, J. Keshet, and D. McAllester. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, 2010.

[39] A. Herschtal and B. Raskutti. Optimising area under the ROC curve using gradient descent. In *Proceedings of the International Conference on Machine Learning*, 2004.

[40] M. Hosseini and S. Lee. Learning sparse gaussian graphical models with overlapping blocks. In *Advances in Neural Information Processing Systems*, 2016.

[41] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 2002.

[42] Y. Jacob, L. Denoyer, and P. Gallinari. Learning latent representations of nodes for classifying in heterogeneous social networks. In *ACM International Conference on Web Search and Data Mining*, 2014.

[43] M Jaggi, S Lacoste-Julien, M Schmidt, and P Pletscher. Block-coordinate frank-wolfe for structural svms. In *Proceedings of the International Conference on Machine Learning*, 2012.

[44] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the International Conference on Machine Learning*, 2013.

[45] T. Joachims. A support vector method for multivariate performance measures. In *Proceedings of the International Conference on Machine Learning*, 2005.

[46] T. Joachims. A support vector method for multivariate performance measures. In *Proceedings of the International Conference on Machine learning*, 2005.

[47] T. Joachims, T. Finley, and C. Yu. Cutting-plane training for structural SVMs. *The Journal of Machine Learning Research*, 2009.

[48] T. Joachims, T. Finley, and C. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 2009.

[49] H. Kar, P.and Narasimhan and P. Jain. Online and stochastic gradient methods for non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, 2014.

[50] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the ACM Symposium on Theory of Computing*, 1984.

[51] D. Kim. Minimizing structural risk on decision tree classification. In *Multi-Objective Machine Learning*. Springer, 2006.

[52] D. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

[53] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[54] J. Kivinen and M. Warmuth. Relative loss bounds for multidimensional regression problems. *The Journal of Machine Learning Research*, 2001.

[55] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM*, 2002.

[56] A. Koster, S. Van Hoesel, and A. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 1998.

[57] A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

[58] M. P. Kumar and P. Dokania. Rounding-based moves for semi-metric labeling. *The Journal of Machine Learning Research*, 2016.

[59] M. P. Kumar, P. Torr, and A. Zisserman. Solving markov random fields using second order cone programming relaxations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, 2006.

[60] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. *Machine learning*, 2002.

[61] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 1998.

[62] S. Maji, L. Bourdev, and J. Malik. Action recognition from a distributed representation of pose and appearance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[63] S. Maji, L. Bourdev, and J. Malik. Action recognition from a distributed representation of pose and appearance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[64] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the ACL Conference on Natural Language Learning*, 2002.

[65] A. Mishra, K. Alahari, and C. V. Jawahar. Scene text recognition using higher order language priors. In *Proceedings of the British Machine Vision Conference*, 2012.

[66] P. Mohapatra, P.K. Dokania, C.V. Jawahar, and M. Pawan Kumar. Partial linearization based optimization for multi-class svm. In *Proceedings of the European Conference on Computer Vision*, 2016.

[67] P. Mohapatra, C.V. Jawahar, and M. P. Kumar. Learning to round for discrete labeling problems. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2018.

[68] P. Mohapatra, C.V. Jawahar, and M. Pawan Kumar. Efficient optimization for average precision svm. In *Advances in Neural Information Processing Systems*, 2014.

[69] P. Mohapatra, M. Rolinek, C.V. Jawahar, Kolmogorov V., and M. P. Kumar. Efficient optimization for rank-based loss functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[70] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach: a case study in intensive care monitoring. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1999.

[71] M. Muramatsu and T. Suzuki. A new second-order cone programming relaxation for max-cut problems. *Journal of the Operations Research Society of Japan*, 2003.

[72] H. Narasimhan. Learning with complex loss functions and constraints. In *International Conference on Artificial Intelligence and Statistics*, 2018.

[73] H. Narasimhan and S. Agarwal. A structural svm based approach for optimizing partial auc. In *International Conference on Machine Learning*, 2013.

[74] H. Narasimhan, A. Cotter, and M. Gupta. Optimizing generalized rate metrics with three players. In *Advances in Neural Information Processing Systems*, 2019.

[75] H. Narasimhan, P. Kar, and P. Jain. Optimizing non-decomposable performance measures: A tale of two classes. In *International Conference on Machine Learning*, 2015.

[76] G. Papandreou, A. Yuille, S. Nowozin, P Gehler, J. Jancsary, and C. Lampert. Advanced structured prediction. 2014.

[77] M Patriksson. Partial linearization methods in nonlinear programming. *Journal of Optimization Theory and Applications, Springer, 1993*, 1993.

[78] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems*, 2000.

[79] J. Quinlan. Classification and regression trees. *Programs for Machine Learning*, 2011.

[80] P. Ravikumar and J. Lafferty. Quadratic programming relaxations for metric labeling and markov random field map estimation. In *Proceedings of the International Conference on Machine Learning*, 2006.

[81] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 2011.

[82] C. Shen, H. Li, and N. Barnes. Totally corrective boosting for regularized risk minimization. *arXiv preprint arXiv:1008.5188*, 2010.

[83] T. Shi, J. Steinhardt, and P. Liang. Learning where to sample in structured prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2015.

[84] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *The International Journal of Computer Vision*, 2009.

[85] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*, 2015.

[86] Y. Song, A. Schwing, R. Zemel, and R. Urtasun. Training deep neural networks via direct loss minimization. In *Proceedings of the International Conference on Machine Learning*, 2016.

[87] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. Mit Press, 2012.

[88] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, 2013.

[89] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[90] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, 2003.

[91] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*, 2004.

[92] J. Thapper and S. Živný. The complexity of finite-valued CSPs. *Journal of the ACM*, 2016.

[93] I. Tsochantaridis, T. Hofmann, Y. Altun, and T. Joachims. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*, 2004.

[94] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 2005.

[95] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *The International Journal of Computer Vision*, 2013.

[96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM review*, 1996.

[97] V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[98] M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolínek. Differentiation of blackbox combinatorial solvers. In *Proceedings of the International Conference on Learning Representations*, 2020.

[99] J. Vondrák, C. Chekuri, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the ACM Symposium on Theory of Computing*, 2011.

[100] M. J. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008.

[101] R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 1991.

[102] D. Williamson and D. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[103] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

[104] Y. Yuan, X. Chen, and J. Wang. Object-contextual representations for semantic segmentation. In *Proceedings of the European Conference on Computer Vision*, 2020.

[105] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[106] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[107] C. Zhang, B. Shahbaba, and H. Zhao. Hamiltonian monte carlo acceleration using surrogate functions with random bases. *Statistics and Computing*, 2015.

[108] X. Zhang, A. Saha, and S. V. N. Vishwanathan. Accelerated training of max-margin markov networks with kernels. *Theoretical Computer Science*, 2014.