Adaptive Probabilities in Stochastic Optimization Algorithms

Master Thesis
Lei Zhong
April 10, 2015

Advisors: Prof. Dr. Thomas Hofmann, Dr. Martin Jaggi
Department of Computer Science, ETH Zürich
Abstract

Stochastic optimization methods have been extensively studied in recent years. In some classification scenarios such as text document categorization, unbiased methods such as uniform sampling have negative effects on the convergence rate, because of the effects of the potential outlier data points on the estimator. Consequently, it would take more iterations to converge to the optimal value for uniform sampling than non-uniform sampling, which normally allocates larger probability to samples with larger norm. Based on the intuition of non-uniform sampling, we investigate adaptive sampling in which we change the probability per iteration or per epoch during the execution of the algorithm.

In this thesis, we start by introducing non-uniform sampling with Stochastic Gradient Descent (SGD), and then study the impact of different choices of stepsize on convergence rate as well as derive theorems for the convergence rate. We analyze Stochastic Dual Coordinate Ascent (SDCA) and obtain the upper bound on the number of iterations to reach a specific level of optimality. This is followed by investigation of adaptive strategies which change the probabilities according to some historical information. We provide theoretical analysis to prove that they further reduce the number of iterations needed to reach the same level of optimality. The adaptive variants of SGD and SDCA are discussed and compared to the existing counterparts. The results show that both AdaSGD and AdaSDCA outperform the existing models. We also provide two online variants which update the probabilities immediately to perform adaptive sampling without the need of additional passes through the data.

Keywords. Adaptive Sampling, Non-Uniform Sampling, Stochastic Gradient Descent, Stochastic Dual Coordinate Ascent
Acknowledgments

I would like to thank my supervisor Dr. Martin Jaggi and Prof. Thomas Hofmann for giving me this opportunity to do a master thesis on this challenging but wonderful project. Thanks all the people in Data Analytics Lab for listening and giving suggestions to my experiments and report.

Further, I would like to offer special thanks to Ziyi and Daniel for constructive feedback and proofreading my report.

Lei Zhong
April. 10, 2015


5 Towards Adaptive Non-Uniform Sampling

5.1 Adaptive Stochastic Gradient Descent
5.1.1 Idea behind SGD
5.1.2 Adaptive Sampling for SGD

5.2 Adaptive Stochastic Variance Reduced Gradient Descent
5.2.1 Idea behind SVRG

5.3 Adaptive Stochastic Dual Coordinate Ascent
5.3.1 Idea behind SDCA
5.3.2 Using Duality Gaps to Update the Probabilities
5.3.3 Using Subgradients to Update the Probabilities

5.4 Comparison with Existing Adaptive Algorithms
5.4.1 AdaGrad
5.4.2 AdaSDCA+

6 Online Adaptive Non-Uniform Sampling

6.1 Online Adaptive Stochastic Gradient Descent
6.2 Online Adaptive Stochastic Dual Coordinate Ascent

7 Discussions

7.1 Setting Probability
7.1.1 Minimum Probability
7.1.2 Initial Probability

7.2 Sampling Methods
7.2.1 Naive Approach
7.2.2 Binary Search
7.2.3 Segment Tree

7.3 Comparison of Complexity
7.3.1 Sampling and Probability Update
7.3.2 Total Computational Cost

8 Experimental Results

8.1 Setup
8.1.1 Training
8.1.2 Finding the Regularization Parameter $\lambda$ which Minimizes Test Error

8.2 Non-Uniform Probability
8.2.1 Initial Probabilities for Non-Uniform Sampling
8.2.2 Verifying the Convergence of Duality Gap
8.2.3 Comparison of Uniform Sampling with Non-Uniform Sampling

8.3 Adaptive Probabilities
8.3.1 Verifying the Alignment of SGDs and SDCAs
8.3.2 Histogram of Probability Changes
8.3.3 Performance of Different Values of $k$
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.4 Performance of Two Updating Algorithms</td>
<td>62</td>
</tr>
<tr>
<td>8.3.5 Different Adaptive Strategies for AdaSDCA</td>
<td>63</td>
</tr>
<tr>
<td>8.3.6 Comparison of Adaptive Algorithms</td>
<td>63</td>
</tr>
<tr>
<td>8.3.7 Comparison of Adaptive Algorithms with Non-Adaptive Algorithms</td>
<td>64</td>
</tr>
<tr>
<td>8.4 Online Algorithms</td>
<td>67</td>
</tr>
<tr>
<td>8.4.1 Online Adaptive Algorithms vs. Non-Adaptive Algorithms</td>
<td>67</td>
</tr>
<tr>
<td>8.5 Summary</td>
<td>67</td>
</tr>
<tr>
<td>9 Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>A Segment Tree</td>
<td>71</td>
</tr>
<tr>
<td>B Implementation Details</td>
<td>75</td>
</tr>
<tr>
<td>C Glossary</td>
<td>77</td>
</tr>
<tr>
<td>Bibliography</td>
<td>79</td>
</tr>
</tbody>
</table>
List of Figures

8.1 Comparison of uniform sampling with non-uniform sampling on astro-ph ........................................ 57
8.2 Histogram of sampling probabilities changed by AdaSGD in selected epochs on rcv1 .................................. 58
8.3 Histogram of sampling probabilities changed by AdaSDCA in selected epochs on rcv1 .................................. 59
8.4 Histogram of sampling probabilities changed by AdaSGD in selected epochs on astro-ph ................................ 60
8.5 Histogram of sampling probabilities changed by AdaSDCA in selected epochs on astro-ph ................................ 61
8.6 Effect of different values of k on rcv1 ............................................. 63
8.7 Comparison of two updating algorithms for AdaSGD and AdaSDCA on rcv1 ........................................... 63
8.8 Comparison of AdaSDCA and AdaSDCAS on rcv1 ...................................... 64
8.9 Comparison of five adaptive algorithms on rcv1 .................................. 64
8.10 Comparison of three adaptive algorithms on astro-ph .................................. 65
8.11 Comparison of the total running time to reach the same optimality ........................................... 66
8.12 Comparison of the vector operations taken to reach the same optimality ........................................... 66
8.13 Comparison of adaptive algorithms with non-adaptive algorithms on rcv1 .................................................. 67
8.14 Comparison of adaptive algorithms with non-adaptive algorithms on astro-ph .......................................................... 67
8.15 Comparison of online adaptive algorithms with non-uniform algorithms on rcv1 ........................................... 68
8.16 Comparison of online adaptive algorithms with non-uniform algorithms on astro-ph ........................................... 68
List of Tables

7.1 Computational cost of one epoch and properties of different algorithms ........................................... 51
8.1 Datasets for empirical study .................................................................................................................. 55
8.2 Test error with different values of $\lambda$ on dataset rcv1 ............................................................... 55
8.3 Test error with different values of $\lambda$ on dataset astro-ph .......................................................... 55
8.4 Average duality gap at different epochs on rcv1 for $\lambda = 0.001$ ...................................................... 56
8.5 Average duality gap at different epochs on astro-ph for $\lambda = 0.001$ .............................................. 56
8.6 Pairwise alignment of SGDs and SDCAs on rcv1 ................................................................................. 62
8.7 Detailed training time and total running time per epoch on rcv1 ...................................................... 65
8.8 Detailed training time and total running time per epoch on astro-ph .............................................. 65
8.9 The number of epochs taken to reach the same level of optimality .................................................. 66
Chapter 1

Introduction

Stochastic Optimization [15, 28, 30, 32, 35] is the process of minimizing or maximizing an objective function under randomness in a parameter. It has been extensively studied in recent years. Stochastic optimization is particularly well suited for problems where there is random noise in measurements or where random choice is made. In addition to its broad application in most real problems where perfect information is not available, the computational efficiency it provides also contributes to its popularity.

Stochastic optimization problem is difficult to interpret for some data samples with extremely peculiar distributions. In order to solve this type of problem we can represent the original objective function as the arithmetic mean of partial objective functions. By introducing the concept of partial objective, we can formally define minimization of the partial objective function as Empirical Risk Minimization. This formulation includes for example Support Vector Machines (SVMs), Logistic Regression, Ridge Regression, etc..

There are two types of approaches to solve this kind of formulation. These approaches are online algorithms in the sense that they process only one data example per iteration.

One is called Stochastic (Sub)Gradient Descent (SGD) and the other is Stochastic Dual Coordinate Ascent (SDCA). A sample for them is a data point selected from a fixed set of data points. At each iteration, SGD selects a sample according to the probability distribution and updates a primal variable by a gradient step with the partial (stochastic) gradient vector obtained from the sample. On the other hand, SDCA similarly selects a sample according to the probability distribution and updates the primal variable by the corresponding dual variable, which is updated by calculating the increment by some equation. We give the detailed explanation about primal, dual variables and the algorithms in Chapter 4.
1. Introduction

We now briefly introduce the related literature for the topic of this thesis, and explain why we study these specific topics in greater depth. Here we only give a short overview, but details in Chapter 2.

Stochastic Gradient Descent algorithms have always been studied to solve the optimization problem Support Vector Machine (SVM). Pegasos by Shalev-Schwartz et al. (2010) [27] was the first paper to have a running time independent of the size of the training set (for a linear kernel). This property makes it particularly well suited for learning from large datasets. However, it bounds the convergence rate by maximum norm of stochastic subgradients instead of the expectation of the subgradients. This will result in a much weaker convergence rate in the presence of outliers. The logarithmic term in the convergence rate of Pegasos’ convergence rate is eliminated by Lacoste-Julien et al. (2012) [15]. They apply the weighted average technique for each iterate on (projected\(^1\)) stochastic gradient descent.

Likewise, Takáč et al. (2013) [31] also shows that the spectral norm of the data affects the parallelization speedup for both SGD and SDCA. In their paper, they discussed the performance of using mini-batch in stochastic optimization of SVMs. Mini-batch uses several training points at each iteration rather than only a single one and computes the update based on each point individually and then combines them. This method, however, empirically yields no improvement in convergence rate.

Up to that point, most authors have focused on uniform sampling because it guarantees the unbiasedness of the estimate. Nonetheless, a noticeably high variance acquired by using uniform sampling exerts negative influence on convergence of the optimization procedure. However, Zhao and Zhang (2014) [35] propose that stochastic optimization with importance sampling improves convergence rate by decreasing stochastic variance. They focused on proximal SGD and proximal SDCA and provided theoretical analysis proving a significant improvement in the convergence rate.

In our paper, we will adopt the new technique from Lacoste-Julien et al. (2012) [15] to extend and obtain an \(O(\frac{1}{T})\) convergence rate for non-uniform sampling. Unlike Zhao and Zhang (2014) [35], we show in this thesis that the convergence rate can still be improved by adaptively changing probabilities and explain how importance sampling is used with plain SGD and SDCA.

Finally, adaptive methods are widely used for online learning and stochastic optimization. AdaGrad by Duchi et al. (2011) [4] carries out the adaptation based on the knowledge obtained from previous iterations and enhances the informativeness of the learning procedure. The method described in this paper performs adaptive modification on the proximal function, simplifying

\(^1\)Projected SGD bounds the norm of solution by some constant.
setting a learning rate and choosing the best proximal function. When applied to datasets with sparse features, AdaGrad has its drawback because the running time depends on the number of features and the size of dataset. Csiba, Zheng and Richtárik (2015) [3] are the first ones to introduce Csiba-AdaSDCA, an adaptive variant of SDCA solving empirical risk minimization problems which is similar to the work in this thesis. However, there are some flaws including that Csiba-AdaSDCA is unrealistic in terms of running time complexity. The heuristic version Csiba-AdaSDCA+ lacks supporting theorems and proofs. Besides, it is not well-informed how to set the parameter \( m \) which is used for scaling the probability in Csiba-AdaSDCA+.

We present an adaptive SDCA that overcomes limitations of Csiba (2015) [3], gives more in-depth analysis and sufficient experiments to verify theoretical statements. We also present an adaptive SGD that has both running time and performance advantage over its counterparts including AdaGrad.

1.1 Uniform vs. Non-Uniform

Sampling deals with the selection of a subset from all the samples to estimate characteristics of the the whole sample set. In the context of SGD and SDCA, sampling is concerned with selecting one training sample or one dual coordinate from the training data and updating the model parameter by using the selected sample or dual coordinate. If the sample or the dual coordinate is selected uniformly at random, then the algorithm uses uniform sampling. If the selection is implemented according to some probability distribution, then the sampling is non-uniform.

In traditional stochastic optimization, uniform sampling is extensively used because it yields an unbiased estimate of the true quantity. The sampled values, nevertheless, can vary significantly and thus results in rather large variance. This is shown in the paper of Zhao et al. (2014) [35]. To reduce variance and hence improve convergence rate, we consider using non-uniform sampling.

To generate probabilities for non-uniform sampling, we adopt different strategies for known distribution of samples: proportional to the norm of samples, inverse proportional to the norm of samples and proportional to the norm of sample plus some constant. The advantage of using non-uniform sampling is to change the bound of the convergence rate from the maximum of some quantities of samples to the mean. We explain it further in Chapter 4.

1.2 Adaptivity

Adaptivity means that the model parameters are adjusted automatically (i.e. probability) based on the information available thus far. Adaptivity is par-
1. Introduction

particularly useful for online algorithms which process their input one-by-one in a serial manner without having the entire input right from the start. Since the data arrives one at a time, the adjustment of the model parameters can be made according to the newly arrived sample such that the prediction of the future samples is more precise.

Adaptive variants of SGD includes Adaptive Encoding [19] and Adaptive Coordinate Frequencies (ACF) [8]. Adaptive SDCA [3] and adaptive subgradient methods [4] are also present. They are only a minority when compared to vast number of works on the extensively studied SGD and SDCA. We focus on adaptivity in these two algorithms in Chapter 5.

1.3 Online

Online algorithms are used when there is no support for storage of additional information and time for additional computing. Here we want to study online variants of the adaptive algorithms, that is to find schemes to update the sampling probabilities without ever needing additional full passes through the dataset. This is discussed in Chapter 6.

1.4 Contribution

We now briefly highlight the main contributions of this work.

- To the best of our knowledge, we are the first to study non-uniform and adaptive sampling strategies for both SGD and SDCA.
- We provide novel analysis of convergence rate which is applied for non-uniform sampling and derive another theorem for the rate that holds without the logarithmic term.
- We provide several adaptive algorithms and perform novel analysis on the convergence rate for them.
- We introduce online adaptive variants without an additional pass after each epoch.
- Experimental results show that our main adaptive algorithms (AdaSGD and AdaSDCA) perform significantly better than the competition and non-adaptive algorithms.

The rest of the thesis is organized as follows. Chapter 2 reviews the related work. In Chapter 3, we present several mathematical preliminaries that we will be using throughout the thesis. We study and analyze non-uniform sampling in Chapter 4, adaptive sampling in Chapter 5 and online variants in Chapter 6. In Chapter 7, we will discuss how to set probability, compare the sampling methods and complexity. Chapter 8 provides experimental
1.4. Contribution

results and comparison of the performance of the mentioned algorithms. Chapter 9 concludes the thesis.
Chapter 2

Related Work

In this chapter, we review the existing literature on stochastic optimization algorithms for typical machine learning problems of the form of regularized loss minimization, such as classification and regression. The two main algorithms are Stochastic Gradient Descent (SGD) and Stochastic Dual Coordinate Ascent (SDCA).

SGD first appeared in the work of Robbins et al. (1951) [26] where it was described as stochastic approximation method back then. They presented a method to solve a root finding problem by successive approximation of the random variables whose initial values are chosen arbitrarily.

Pegasos established by Shalev-Schwartz et al. (2010) [27] was the first paper giving a convergence rate for SGD where the number of iterations necessary to reach a specified accuracy was shown to be independent of the size of dataset. Thereby, this result showed strong advantages for SGD as compared to existing batch optimization methods applied to the same task. Later, Johnson et al. (2013) [12] and Lin (2014) [32] introduce a technique called ‘Variance Reduction’ on SGD to speed up the process. However, this method needs a large amount of additional computations.

There are also several papers focusing on mini-batch variants. Richtárik et al. (2014) [13] establish a mini-batch semi-SGD called mS2GD which is used to solve the minimization problem of a strongly convex composite function. This process consists of a deterministic step (for variance reduction) and many stochastic steps afterwards. Mini-batching is applied to the computation of each step by sampling several instead of only one functions and computing their gradients. Shalev-Shwartz et al. (2013) [29] give an accelerated mini-batch version of SDCA that solves regularized loss minimization problems and proves its fast convergence rate. Takac et al. (2013) [31] derive both mini-batch primal (SGD) and dual (SDCA) methods for SVMs with convergence constants depending on the spectral norm of the data.
2. Related Work

Stochastic Dual Coordinate Ascent (SDCA) was first introduced in the work of Hsieh et al. (2008) [9] which presents a method for linear SVMs with L1- and L2 loss functions. This method is especially well suited for dealing with large-scale sparse data. Experiments show that it is much faster than its counterparts including Pegasos [27], TRON [16], and SVM$^{\text{perf}}$ [11].

SDCA is extensively studied by Zhang et al. (2013) [30] which justifies that SDCA has comparable or even better theoretical guarantees than SGD for regularized loss minimization and demonstrates its effectiveness in practice.

There are also some papers considering proximal SGD and proximal SDCA. Zhang et al. (2012) [28] presents a proximal version of SDCA for many regularized loss minimization problems and obtains a comparable or sometimes better result in convergence rates than the state-of-the-art methods. Lin et al. (2014) [17] introduce an accelerated proximal coordinate gradient method for regularized empirical risk minimization which achieves an improved convergence rate compared to other existing randomized proximal coordinate gradient methods or SDCA methods under different convexity conditions.

Other related papers include S2CD [14], [25] and [22]. Qu et al. (2014) [14] introduce a semi-stochastic coordinate descent (S2CD) method for minimization of a strongly convex function. This method is composed of a first deterministic step followed by numerous stochastic steps where a random function and a random coordinate are picked using nonuniform distributions at each step for the update of a single coordinate. Richtárik et al. (2013) [25] propose and study a parallel coordinate descent method where a non-uniformly chosen random subset of coordinates is updated in parallel. They discuss the assignment of probabilities to optimize the bound on convergence rates under a strong convexity condition. In the case of huge-scale optimization problems, Nesterov (2010) [22] proposes coordinate descent methods based on random partial update of decision variables for which he proves global estimates of the convergence rate and implements constrained as well as unconstrained versions of objective functions.

Distributed stochastic optimization is also a currently growing research field. Bradley et al. (2011) [2] propose a coordinate descent algorithm that minimizes L1-regularized loss in parallel setting. They prove convergence bounds which accelerates the algorithm in a near-linear manner up to a problem-dependent limit.

Distributed algorithms are particularly well suited in dealing with Big Data, which is a major topic in machine learning and optimization research. Richtárik et al. (2012) [23] show that parallelization can speed up randomized block coordinate descent method when applied to Big Data optimization. Theoretically, the speedup depends on the number of parallel processors and separability of the smooth component of the objective function. A hybrid coordinate descent method for solving loss minimization problems with big
data was studied in the paper of Richtárik et al. (2013) [24]. The coordinates are partitioned between different compute nodes, and their selection and updates are carried out independently and in parallel. Fercoq et al. (2014, 2013, 2013) [5, 6, 7] propose and analyze different versions of proximal coordinate descent algorithms under various settings. [7] involves an efficient distributed randomized coordinate descent method for minimizing regularized non-strongly convex loss function and studies the stepsize parameters theoretically. [6] proposes a simultaneously accelerated, parallel and proximal stochastic coordinate descent method for minimization of the sum of convex functions. [5] discusses properties of a family of randomized parallel coordinate descent methods for minimizing the sum of non-smooth and separable convex functions. They prove mathematically that fewer iterations are needed under the presence of more processors.

Two papers of Yang et al. (2013) [33, 34] focus on distributed SDCA. [33] presents a distributed SDCA algorithm in a star network, analyzing the tradeoff between computation and communication. This algorithm has competitive performance in terms of running time of SVMs as compared to other distributed algorithms including SGD and ADMM. [34] presents theoretical analysis of the convergence rates of the distributed SDCA algorithm and confirms its better performance in comparison with the plain variant. Liu et al. (2014) [18] introduce an asynchronous parallel stochastic proximal coordinate descent algorithm for minimizing a composite objective function and prove its linear convergence rate under the assumption of strong convexity. Distributed block coordinate descent methods by Mahajan et al. (2014) [20] and Richtárik et al. (2014) [21] are designed for training L1-regularized linear classifiers and minimizing partially separable functions respectively. CoCoA by Jaggi et al. (2014) [10] is a distributed optimization algorithm for large-scale machine learning which uses local computation in a primal-dual setting to decrease the amount of communication required. They provide a strong convergence rate analysis showing CoCoA converges more quickly while reaching the same accuracy of solution in comparison to state-of-the-art mini-batch versions of SGD and SDCA.

Adaptive coordinate descent is an extension of the coordinate descent algorithm. Sebag et al. (2011) [19] use Adaptive Encode (AE) that transforms the coordinate system such that the new coordinates are as uncorrelated as possible with regard to the objective function, which is then solved by a coordinate descent algorithm, showing comparable performance to other state-of-the-art variants.

Glassmakers et al. (2013) [8] propose an adaptive coordinate frequencies method where the coordinates are not treated equally and the relative frequencies of coordinates are subject to online adaptation. AdaGrad by Duchi et al. (2011) [4] and Csiba-AdaSDCA by Csiba et al. (2015) [3] change the proba-
2. Related Work

bility distribution during the running of the algorithms and hence are most related to our work. [4] proposes subgradient methods that dynamically integrate knowledge of data obtained in previous iterations for a more informative gradient-based learning. [3] presents Csiba-AdaSDCA+ which is an adaptive version of SDCA that solves the regularized empirical risk minimization problem. This method changes the probability adaptively over the dual variables at each iteration of the process. However, neither do they perform better than Adaptive SDCA in this thesis, nor do they provide theoretical analysis on the convergence rate.
In this chapter, we guide the reader through the mathematical foundations of the problems of interest in this thesis.

### 3.1 Mathematical Foundations

In this thesis, our main motivating example is binary classification with hinge loss, which is the reason why we focus on the Support Vector Machine (SVM) problem. However, our algorithms are generally applicable to other machine learning algorithms, as long as the Remark 3.1 holds.

In machine learning, many models of classification and regression analysis are of the following form. Given a training set \((x_i, y_i)_{i=1}^n\) with features \(x_i \in \mathbb{R}^d\) and labels \(y_i \in \mathbb{R}\), we want to find a solution to minimize the risk. General linear models for risk minimization problems in machine learning can be formulated as the task of minimizing the following objective function. An important observation is that the objective function can be written as a sum of \(n\) terms, each depending on only one of the training datapoints.

\[
f(w) := \ell(w) + \lambda r(w)
\]

where

\[
\ell(w) := \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, w, y_i).
\]

and

\[
r(w) := \frac{1}{2} \|w\|_2^2
\]

Here, \(\ell(., y_i) : \mathbb{R} \to \mathbb{R}\) is a loss function and \(r(\cdot)\) takes the role of a regularizer.

For example, in the case of SVM, the training labels take values \(y_i \in \{-1, +1\}\), and \(\ell(\cdot)\) is typically chosen as the hinge loss function.
3. Preliminaries

Remark 3.1 In the whole thesis, we will assume that \( f \) is a convex function, and the same for \( \ell(.,y_i) \) and \( r(.) \).

3.1.1 Regularizers

**L2 regularization**: The simplest regularization is the squared L2 regularization
\[
r(w) := \frac{1}{2}\|w\|_2^2.
\]
As we will see from Equation (4.10), it is a 1-strongly convex regularization function.

**L1 regularization**: The L1 regularizer is widely used in applications such as Lasso. However, the L1 regularizer is not a strongly convex function. This can be solved by adding a L2 regularizer. The resulting regularizer is
\[
r(w) := \frac{1}{2}\|w\|^2 + \frac{\sigma}{\lambda}\|w\|_1
\]
where \( \sigma \in \mathbb{R}_+ \).

3.1.2 Loss Functions

**Hinge loss**: \( \ell(a) = [1-a]_+ := \max\{0,1-a\} \).

**Smoothed hinge loss**: Smoothed hinge loss is parameterized by a scalar \( \gamma \leq 0 \) and is defined as:
\[
\ell_{\gamma}(a) := \begin{cases}
0, & a \geq 1 \\
1 - a - \gamma/2, & a \leq 1 - \gamma \\
\frac{1}{2\gamma}(1-a)^2 & \text{o.w.}
\end{cases}
\]

**Squared loss**: The loss function is defined as
\[
\ell(a) := \frac{1}{2}(a - y)^2
\]

In the rest of this thesis, we use \( \|\cdot\| \) to denote L2-norm for simplicity.

3.2 Definitions

Here we present the definitions that we will use throughout the thesis.

**Definition 3.2** A function \( h: \mathbb{R}^d \to \mathbb{R} \) is \( \lambda \)-strongly convex, if for all \( u, v \in \mathbb{R}^d \), we have
\[
h(u) \geq h(v) + \nabla h(v)^T (u - v) + \frac{\lambda}{2}\|u - v\|^2,
\]
or equivalently [35], ∀s ∈ [0, 1]
\[ h(su + (1 - s)v) \leq sh(u) + (1 - s)h(v) - \frac{\lambda s(1 - s)}{2} \|u - v\|^2. \]

Note that for the choice of λ = 0 for the strong convexity parameter, the above definition becomes equivalent to convexity.

**Definition 3.3** A function h: \( \mathbb{R}^d \rightarrow \mathbb{R} \) is \( L \)-Lipschitz, if for all \( u, v \in \mathbb{R}^d \), we have
\[ |h(u) - h(v)| \leq L \|u - v\|, \]
where \( \| \cdot \| \) is a norm.

**Definition 3.4** A function h : \( \mathbb{R}^d \rightarrow \mathbb{R} \) is \( (1/\gamma) \)-smooth, if it is differentiable and its gradient is \( (1/\gamma) \)-Lipschitz, or equivalently for all \( u, v \in \mathbb{R}^d \), we have
\[ h(u) \leq h(v) + \nabla h(v)^\top (u - v) + \frac{1}{2\gamma} \|u - v\|^2. \]

### 3.3 Applications

In this section, we list other machine learning scenarios where our algorithms can be applied to.

#### 3.3.1 Ridge Regression

Ridge Regression uses a squared loss, which is shown as following
\[ \ell(w, x_i, y_i) := \frac{1}{2} (\langle w, x_i \rangle - y_i)^2 \]
and
\[ r(w) := \frac{1}{2} \|w\|^2. \]

#### 3.3.2 LASSO

LASSO (Least Absolute Shrinkage and Selection Operator) uses a squared loss and L1 regularizer. It minimizes the sum of square loss with a penalty (coefficient with sum of absolute values of the estimator). The resulting estimator is generally sparse which is widely used as automatic feature selection.
\[ \ell(w, x_i, y_i) := \frac{1}{2} (\langle w, x_i \rangle - y_i)^2 \]
and
\[ r(w) := \|w\|_1. \]

For more details, the reader can refer to Bishop et al. (2006) [1].
In this chapter, we start by introducing Stochastic Gradient Descent. Following the basic SGD (Pegasos) algorithm as described in [27], the initial weight vector \( w \) for the algorithm is set to be 0. In each iteration, we pick a random training sample \((x_i, y_i)\), with probability \( p_i \).

While the original SGD algorithm was defined using uniform sampling probabilities \( p_i := 1/n \) for each \( i \), we will here generalize this by allowing arbitrary non-uniform sampling probabilities given by \( p_i \geq 0 \) with \( \sum_{i=1}^{n} p_i = 1 \).

After the sampling step, the algorithms consider the \( i \)-th partial objective function as follows:

\[
f(w, i) := \ell(\langle w, x_i \rangle, y_i) + \lambda r(w). \tag{4.1}
\]

Since hinge loss is not differentiable when \( \langle w, x_i \rangle y_i = 1 \), we consider the subgradient here. The subgradient of the partial objective \( f(w^t, i_t) \) at the \( t \)-th iteration — where \( i_t \) is chosen with probability \( p_{i_t} \) — is given by:

\[
\chi_{i_t}^t(w^t) := \nabla f(w^t, i_t) = \nabla \ell_{i_t}(w^t) + \lambda \nabla r(w^t), \tag{4.2}
\]

which is equal to

\[
\chi_{i_t}^t(w^t) := \ell'(\langle w^t, x_{i_t} \rangle, y_{i_t}) x_{i_t} + \lambda \nabla r(w^t)
\]

where \( \ell'(u, y) \) denotes a subgradient with respect to the first variable \( u \).

### 4.1 Non-Uniform Stochastic Gradient Descent

Each iteration of the stochastic gradient descent (SGD) algorithm is defined by sampling a random index \( i \) with probability \( p_i \). The iterate is then up-
4. Stochastic Optimization with Non-Uniform Sampling

dated as

\[ w^{t+1} := w^t - \eta_t g^t_i \]  

(4.3)

It is easy to see that our update direction — the stochastic subgradient \( g^t_i \) — can be made an unbiased estimator of the true (full) subgradient \( \nabla f(w) \). In other words, \( \mathbb{E}[g^t_i \mid w^t] \) is almost surely a subgradient of \( f \). To achieve this, we define

\[ g^t_i := g^t_i(w^t) := \frac{\chi^t_i(w^t)}{np_i}, \]  

(4.4)

which gives

\[ \mathbb{E}[g^t_i \mid w^t] := \mathbb{E} \left[ \frac{\chi^t_i(w^t)}{np_i} \right] = \frac{1}{n} \sum_{i=1}^{n} \chi^t_i(w^t) = \nabla f(w^t), \]

where the expectation is over the random choice of the index \( i \), according to the given probabilities \( p_i \).

4.1.1 Algorithm

A formal definition of the non-uniform SGD algorithm (NonUnifSGD) is given below in Algorithm 1.

**Algorithm 1: NonUnifSGD (Non-Uniform Stochastic Gradient Descent)**

**Input:** \( \lambda > 0 \), probabilities \( p_i = \frac{\|x_i\| + \sqrt{\lambda}}{\sum_{i=1}^{n} \|x_i\| + \sqrt{\lambda}}, \) or \( p_i = \frac{\|x_i\|}{\sum_{i=1}^{n} \|x_i\|}, \forall i \in \{1, \ldots, n\} \).

**Data:** \( \{(x_i, y_i)\}_{i=1}^{n} \).

**Initialize:** \( w^1 = 0 \).

**for** \( t = 1, 2, \ldots, T \)

| Sample \( i_t \) from \( \{1, \ldots, n\} \) based on \( p \);  
| Set stepsize \( \eta_t \leftarrow \frac{1}{\lambda T} \);  
| Calculate \( \ell' \leftarrow \ell'(\langle w^t, x_{i_t} \rangle, y_{i_t}); \)  
| Set \( \chi^t_{i_t}(w^t) \leftarrow \ell'x_{i_t} + \lambda \nabla r(w^t); \)  
| Set \( g^t_{i_t} \leftarrow \frac{\chi^t_{i_t}(w^t)}{np_{i_t}} \);  
| Set \( w^{t+1} \leftarrow w^t - \eta_t g^t_{i_t}; \) |

**end**

**Output:** \( w^{T+1} \)

We denote the unique minimizer of \( f \) by \( w^* \) and we say that a candidate solution \( w \) is \( \varepsilon \)-sub-optimal if

\[ f(w) - f(w^*) \leq \varepsilon. \]
4.1. Non-Uniform Stochastic Gradient Descent

We modify the proof idea of [15] by getting rid of the orthogonal projection of \( w \). In addition, we make the convergence theorem hold for arbitrary probabilities by giving a different definition of \( g_i \). The \( g_i \) in [15] is equivalent to \( \chi_i \) in this thesis, because in the case of uniform sampling \( g_i = \chi_i \). By the definition of the SGD update (4.3), it holds that

\[
\|w^{t+1} - w^*\|^2 = \|w^t - \eta_t g_i^t - w^*\|^2 = \|w^t - w^*\|^2 + \eta_t^2 \|g_i^t\|^2 - 2\eta_t (w^t - w^*)^T g_i^t.
\]

We take the expectation over the sampling probability at time \( t \) on both sides, conditioning on the outcome of the previous step \( w^t \). Note that, based on (4.3) and (4.4), \( w^t \) is independent of \( i_t \) while \( g_i^t \) and \( w^{t+1} \) depend on \( i_t \).

\[
\mathbb{E}[\|w^{t+1} - w^*\|^2 | w^t] = \|w^t - w^*\|^2 + \eta_t^2 \mathbb{E}[\|g_i^t\|^2 | w^t] - 2\eta_t \mathbb{E}[(w^t - w^*)^T g_i^t | w^t] = \|w^t - w^*\|^2 + \eta_t^2 \mathbb{E}[\|g_i^t\|^2 | w^t] - 2\eta_t \nabla f(w^t)(w^t - w^*)
\]

Note that by definition, the objective function \( f \) is \( \lambda \)-strongly convex, therefore

\[
\mathbb{E}[\|w^{t+1} - w^*\|^2 | w^t] \leq \|w^t - w^*\|^2 + \eta_t^2 \mathbb{E}[\|g_i^t\|^2 | w^t] - 2\eta_t [f(w^t) - f(w^*) + \frac{\lambda}{2} \|w^t - w^*\|^2].
\]

After rearranging, we get

\[
f(w^t) - f(w^*) \leq \frac{\eta_t}{2} \mathbb{E}[\|g_i^t\|^2 | w^t] + \frac{1 - \lambda\eta_t}{2\eta_t} \|w^t - w^*\|^2 - \frac{1}{2\eta_t} \mathbb{E}[\|w^{t+1} - w^*\|^2 | w^t].
\]

(4.5)

By taking the expectation again over the random sampling at each step, we have the following bound, which will be the cornerstone of our convergence analysis.

\[
\mathbb{E}[f(w^t)] - f(w^*) \leq \frac{\eta_t}{2} \mathbb{E}[\|g_i^t\|^2] + \frac{1 - \lambda\eta_t}{2\eta_t} \mathbb{E}[\|w^t - w^*\|^2] - \frac{1}{2\eta_t} \mathbb{E}[\|w^{t+1} - w^*\|^2]
\]

(4.6)

4.1.2 Analysis

Here we will prove that our NonUnifSGD (Algorithm 1) has a \( O\left(\frac{\ln T}{T}\right) \) convergence rate.

**Theorem 4.1** Suppose \( f \) is a \( \lambda \)-strongly convex function. If we choose the stepsize \( \eta_t = \frac{1}{\lambda T} \), then after \( T \) iterations of NonUnifSGD (Algorithm 1) starting with \( w^1 = 0 \), it holds that

\[
\mathbb{E}\left[ f\left(\frac{1}{T} \sum_{t=1}^{T} w^t\right) \right] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \mathbb{E}[\|g_i^t\|^2] - \frac{1}{T}
\]

19
where $g_i^t = \frac{x_i^t(w)}{np_i}$ and the expectation is taken with respect to the distribution $p$.

**Proof** With stepsize $\eta_t = \frac{1}{\lambda t}$ plugged into (4.6), we have

$$
\mathbb{E}[f(w^t)] - f(w^*) \leq \frac{1}{2\lambda t} \mathbb{E}[\|g_i^t\|^2] + \frac{\lambda(t - 1)}{2} \mathbb{E}[\|w^t - w^*\|^2] - \frac{\lambda t}{2} \mathbb{E}[\|w^{t+1} - w^*\|^2]$$

(4.7)

We use convexity of the function $f$, as given by Jensen’s inequality:

$$
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{t=1}^{T} w^t \right) \right] - f(w^*) \leq \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[f(w^t)] - f(w^*)
$$

Summing up (4.7) over all steps $t = 1 \ldots T$, we can bound the right hand side of the above inequality

$$
= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[f(w^t)] - f(w^*) \leq \frac{1}{T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g_i^t\|^2]}{2\lambda} - \frac{\lambda}{2} \mathbb{E}[\|w^{T+1} - w^*\|^2]
$$

(where we have used the telescoping sum of the norm terms.)

Re-arranging terms, and trivially bounding the left hand side of Jensen’s inequality by $0 \leq \mathbb{E}[f(\frac{1}{T} \sum_{t=1}^{T} w^t)] - f(w^*)$, we obtain the claimed bound

$$
\mathbb{E}[\|w^{T+1} - w^*\|^2] \leq \frac{1}{\lambda^2 T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g_i^t\|^2]}{t}.
$$

This is a novel result obtained by applying the analysis of Lacoste et al.,[15] to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities. The difference between their analysis and ours is that we keep the $E$ to non-uniform probabilities.

**Definition 4.2** Define $G := \max_{i,t} \{ \|x_i^t(w^t)\|^2 \} \ (i = 1 \ldots n, \ t = 1 \ldots T)$. Define $W := \max_{i,t} \{ \|x_i^t(w^t)\|^2 \} \ (i = 1 \ldots n, \ t = 1 \ldots T)$.

**Corollary 4.3** Suppose $f$ is a $\lambda$-strongly convex function, and assume that according to Definition 4.2, $\max_{i,t} \{ \|x_i^t(w^t)\|^2 \} \leq G$ for all $t$. If we choose $\eta_t = \frac{1}{\lambda t}$ and assume $p_i > \epsilon$ for all $i = \{1, \ldots, n\}$, then from Theorem 4.1,

$$
\mathbb{E}[f \left( \frac{1}{T} \sum_{t=1}^{T} w^t \right)] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g_i^t\|^2]}{t} \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{G}{\epsilon t} \leq \frac{G(\ln T + 1)}{2\lambda \epsilon t}
$$

**Corollary 4.4** Suppose $f$ is a $\lambda$-strongly convex function, and assume that according to Definition 4.2, $\mathbb{E}[\|x_i^t(w^t)\|^2] \leq W$ for all $t$. If we choose $\eta_t = \frac{1}{\lambda t}$ and assume $p_i > \epsilon$ for all $i = \{1, \ldots, n\}$, then from Theorem 4.1,

$$
\mathbb{E}[f \left( \frac{1}{T} \sum_{t=1}^{T} w^t \right)] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g_i^t\|^2]}{t} \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{W}{n^2 \epsilon^2 t} \leq \frac{W(\ln T + 1)}{2\lambda T n^2 \epsilon^2}
$$
While Corollary 4.3 shows that convergence depends on the maximum value of all partial subgradient norms \((G)\), Corollary 4.4 expresses a much more favorable convergence constant, namely the mean of the partial subgradients norms \((W)\). The average of these \(n\) norms is typically much smaller than the maximum value. For example, we can think of a single outlier in the dataset being responsible for the maximum value.

The second part of Theorem 1 from [35] makes a similar claim as Corollary 4.4. In Theorem 4.5 we refine our results to eliminate the logarithmic term in \(T\).

### 4.1.3 Another Analysis

In this subsection, we will use a different stepsize to get a \(O(1/T)\) rate of convergence. That is, we want to eliminate the \(\ln T\) term in the convergence rate. We adopted the idea from [15] to take the weighted average of \(w\) at each iteration to evaluate the gap with function value of \(w^*\). In contrast to [15], our theorem does not bound \(E\|g^t_i\|^2\) for the whole process (see our Corollary 4.6 for bounded version).

**Theorem 4.5** Suppose \(f\) is a \(\lambda\)-strongly convex function. If we choose the step-size \(\eta_t = \frac{2}{\lambda(t+1)}\) [15], then after \(T\) iterations of NonUnifSGD (Algorithm 1) with starting point \(w^1 = 0\), it holds that the weighted average of the iterates satisfies

\[
\mathbb{E}[f\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} tw^t\right)] - f(w^*) \leq \frac{2}{\lambda(T+1)} \max \mathbb{E}[\|g^t_i\|^2]
\]

where \(g^t_i = \frac{\chi^t_i(w^t)}{np_{i}}\), and the expectation is taken with respect to the distribution \(p\).

The proof idea follows Lacoste’s proof idea [15], but we make it more general by allowing non-uniform probabilities.

**Proof** From (4.6) and \(\eta_t = \frac{2}{\lambda(t+1)}\), we have \(\mathbb{E}[f(w^t)] - f(w^*)\)

\[
\leq \frac{1}{\lambda(t+1)} \mathbb{E}[\|g^t_i\|^2] + \frac{\lambda(t-1)}{4} \mathbb{E}[\|w^t - w^*\|^2] - \frac{\lambda(t+1)}{4} \mathbb{E}[\|w^{t+1} - w^*\|^2]
\]

\[
= \frac{1}{t} \left[ \frac{t}{\lambda(t+1)} \mathbb{E}[\|g^t_i\|^2] + \frac{\lambda(t-1)t}{4} \mathbb{E}[\|w^t - w^*\|^2] - \frac{\lambda t(t+1)}{4} \mathbb{E}[\|w^{t+1} - w^*\|^2] \right]
\]

\[
\leq \frac{1}{t} \left[ \frac{1}{\lambda} \mathbb{E}[\|g^t_i\|^2] + \frac{\lambda(t-1)}{4} \mathbb{E}[\|w^t - w^*\|^2] - \frac{\lambda t(t+1)}{4} \mathbb{E}[\|w^{t+1} - w^*\|^2] \right]
\]
As before, we sum up from $t = 1$ to $T$, yielding:

$$
\sum_{t=1}^{T} t \mathbb{E}[f(w^t) - f(w^*)] \leq \frac{1}{\lambda} \sum_{t=1}^{T} \mathbb{E}[\|g_i^t\|^2] + \frac{\lambda}{4} \left[ 0 - T(T+1) \mathbb{E}[\|w^{T+1} - w^*\|^2] \right]
$$

Thus

$$
\mathbb{E}[f\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} tw^t\right)] - f(w^*) \leq \frac{2}{T(T+1)} \sum_{t=1}^{T} t \mathbb{E}[f(w^t) - f(w^*)]
$$

$$
\leq \frac{2}{T(T+1)} \sum_{t=1}^{T} \frac{1}{\lambda} \mathbb{E}[\|g_i^t\|^2] - \frac{\lambda}{2} \mathbb{E}[\|w^{T+1} - w^*\|^2]
$$

The above formula yields

$$
\mathbb{E}[\|w^{T+1} - w^*\|^2] \leq \frac{4}{\lambda^2 T(T+1)} \sum_{t=1}^{T} \mathbb{E}[\|g_i^t\|^2] \leq \frac{4}{\lambda^2 T(T+1)} \max_t \mathbb{E}[\|g_i^t\|^2]
$$

and

$$
\mathbb{E}[f\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} tw^t\right)] - f(w^*) \leq \frac{2}{\lambda T(T+1)} \sum_{t=1}^{T} \mathbb{E}[\|g_i^t\|^2] \leq \frac{2}{\lambda(T+1)} \max_t \mathbb{E}[\|g_i^t\|^2]
$$

Here the convergence rate is $O(1/T)$, compared to $O(\ln T/T)$ in [27, 35].

**Corollary 4.6**  Suppose $f$ is a $\lambda$-strongly convex function, and assume that according to Definition 4.2, $\max t \{\|x_i^t(w^t)\|^2\} \leq G$. If we choose $\eta_t = \frac{2}{\lambda(T+1)}$, then from Theorem 4.5 we have

$$
\mathbb{E}[f\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} tw^t\right)] - f(w^*) \leq \frac{2}{\lambda(T+1)} \max_t \mathbb{E}[\|g_i^t\|^2] \leq \frac{2G}{\lambda n \epsilon(T+1)}.
$$

We bound the convergence rate for non-uniform sampling as inversely proportional to the total time, an improvement on [35, Theorem 1, part 2].

**Corollary 4.7**  Suppose $f$ is a $\lambda$-strongly convex function, and assume that according to Definition 4.2, $\mathbb{E}[\|x_i^t(w^t)\|^2] \leq W$ for all $t$. If we choose $\eta_t = \frac{2}{\lambda(T+1)}$, then from Theorem 4.5 we have

$$
\mathbb{E}[f\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} tw^t\right)] - f(w^*) \leq \frac{2}{\lambda T(T+1)} \sum_{t=1}^{T} \mathbb{E}[\|g_i^t\|^2] \leq \frac{2W}{\lambda(T+1)n^2 \epsilon^2}.
$$
Here again, we make two corollaries by bounding the subgradient in two ways. As written in the Subsection 4.1.2, average subgradient is smaller than maximum subgradient. Therefore, Corollary 4.7 is a tighter bound than Corollary 4.6.

Trivially, it is not necessary to bound the subgradient for arbitrary iterates $t$ and sample $i$. Assume that after a sufficient number of iterations, the subgradient $\|g^t_i\|$ can be bounded within a ‘ball’. Additional iterations will decrease the radius of the ‘ball’. We can then show that $w^t$ approaches $w^*$ much faster in later epochs. This is what inspired us to develop an adaptive algorithm in Chapter 5.

### 4.2 Non-Uniform Stochastic Dual Coordinate Ascent

In this section, we will analyze the convergence rate of an alternative method called Stochastic Dual Coordinate Ascent (SDCA)[31]. Let us consider the dual of (3.1):

$$
\max_{\alpha \in \mathbb{R}^n} D(\alpha) := \frac{1}{n} \sum_{i=1}^{n} -\ell^*_i(-\alpha_i) - \lambda r^*(v(\alpha)).
$$

(4.8)

The relationship between primal variable $w$ and dual variable $\alpha$ is

$$
w(\alpha) := \nabla r^*(v(\alpha)), v(\alpha) := \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i
$$

(4.9)

where $\alpha \in \mathbb{R}^n$.

#### 4.2.1 Analysis

**Proposition 4.8** [35] If $\ell$ is $(1/\gamma)$-smooth with respect to a norm $\| \cdot \|_P$, then its dual function $\ell^*$ is $\gamma$-strongly convex with respect to its dual norm $\| \cdot \|_D$, where

$$
\ell^*(v) := \sup_{w} (v^T w - \ell(w)),
$$

and the dual norm is defined as

$$
\|v\|_D := \sup_{\|w\|_P = 1} v^T w.
$$

In our case, $\| \cdot \|_P = \| \cdot \|_D = \| \cdot \|$. By Proposition 4.8,

$$
r^*(w) := \sup_{x} (w^T x - \|x\|^2/2) = \|w\|^2/2
$$

(4.10)
4. Stochastic Optimization with Non-Uniform Sampling

\[ \mathbf{w}(\alpha) := \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i \mathbf{x}_i. \]

At each iteration, we take a sample \((x_i, y_i)\) from the training set according to their probability \(p\), corresponding to a single dual variable \(\alpha_i = e_i^T \alpha\). At the \(t\)-th step, SDCA will update the dual variable \(\alpha_i^t\) as follows:

\[ \alpha_i^{t+1} := \alpha_i^t + \Delta \alpha_i^t \quad (4.11) \]

\(\Delta \alpha_i^t\) is computed by:

\[ \Delta \alpha_i^t = \arg \max_{\Delta \alpha_i^t} \left[ -\ell^*_i (\alpha_i^t + \Delta \alpha_i^t) - \langle \mathbf{w}^t, \mathbf{x}_i \rangle \Delta \alpha_i - \frac{1}{2 \lambda n} \| \mathbf{x}_i \|^2 \| \Delta \alpha_i \|^2 \right] \quad (4.12) \]

The equation (4.12) is hard to solve [35], so we use another update rule

\[ \Delta \alpha_i^t = s_i (u_i^t - \alpha_i^t) \quad (4.13) \]

where \(s_i > 0\) is the stepsize and \(-u_i^t \in \partial \ell_i(\mathbf{w}^t) \subset \mathbb{R}^d\).

For hinge loss

\[ \ell_i(u_i) = \max(0, 1 - u_i y_i), \quad \ell_i^*(a) = -a y_i \]

with \(a y_i \in [0, 1]\) and \(y_i \in \{ \pm 1 \}\), (4.12) has solution

\[ \Delta \alpha_i = y_i \max(0, \min(1, \frac{1 - y_i x_i^T \mathbf{w}^t}{\| \mathbf{x}_i \|^2 / (\lambda n)} + \alpha_i^t y_i)) - \alpha_i^t. \]

**Lemma 4.9 ([35])** Given a distribution \(p\) at iteration \(t\), if we assume \(l_i\) is \((1 / \gamma_i)\)-smooth in \(\| \cdot \|\) and start with \(\mathbf{w}^1 = 0\) and \(\alpha^1 = 0\), then for any iteration \(t\) and any \(s\) such that \(s_i = \frac{s}{\overline{p_i}} \in [0, 1], \forall i\), we have

\[ \mathbb{E} [D(\alpha^{t+1}) - D(\alpha^t)] \geq \frac{s}{n} [f(\mathbf{w}^t) - D(\alpha^t)] - \frac{s}{2 \lambda n^2} G^t, \quad (4.14) \]

holds for arbitrary \(p\), where

\[ G^t := \frac{1}{n} \sum_{i=1}^{n} (s_i \| \mathbf{x}_i \|^2 - \gamma_i (1 - s_i) \lambda n) (u_i^t - \alpha_i^t)^2, \]

where \(-u_i^t \in \partial \ell_i(\mathbf{w}^t)\).

**Proof** The improvement of the dual objective can be written as

\[ D(\alpha^{t+1}) - D(\alpha^t) = \left[ \frac{1}{n} \sum_{i=1}^{n} -\ell_i^* (-\alpha_i^{t+1}) - \lambda r^* (\frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i^{t+1}) \right] \]

\[ - \sum_{i=1}^{n} A_i \]

24
Therefore
\[
- \left[ \frac{1}{n} \sum_{i=1}^{n} - \ell^*_i (-\alpha^*_i) - \lambda r^* \left( \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha^*_i \right) \right]_{b_i} \sum_{i=1}^{b_i}
\]
By Equation (4.10) and Equation (4.11), we have
\[
A_i := \max_{\Delta \alpha_i} - \ell^*_i (-\alpha^*_i - \Delta \alpha_i) - \frac{\lambda n}{2} \| \omega^t \| + \frac{1}{\lambda n} s_i (u^*_i - \alpha^*_i) x_i \| \|^2
\]
By the definition of the update (4.13), \( \forall s_i \in [0, 1] \)
\[
A_i \geq - \ell^*_i (-\alpha^*_i - s_i (u^*_i - \alpha^*_i)) - \frac{\lambda n}{2} \| \omega^t \| + \frac{1}{\lambda n} s_i (u^*_i - \alpha^*_i) x_i \|^2
\]
\[
= - \ell^*_i (s_i (u^*_i) + (1 - s_i) (-\alpha^*_i)) - \frac{\lambda n}{2} \| \omega^t \| + \frac{1}{\lambda n} s_i (u^*_i - \alpha^*_i) x_i \|^2
\]
Because \( \ell^*_i \) is a \( \gamma_i \)-strongly convex function, we have
\[
A_i \geq - s_i \ell^*_i (-u^*_i) - (1 - s_i) \ell^*_i (-\alpha^*_i) + \frac{\gamma_i}{2} s_i (1 - s_i) (u^*_i - \alpha^*_i)^2
\]
\[
- \frac{\lambda n}{2} \| \omega^t \| + \frac{1}{\lambda n} s_i (u^*_i - \alpha^*_i) x_i \|^2
\]
\[
= - s_i \ell^*_i (-u^*_i) - (1 - s_i) \ell^*_i (-\alpha^*_i) + \frac{\gamma_i}{2} s_i (1 - s_i) (u^*_i - \alpha^*_i)^2
\]
\[
- \frac{\lambda n}{2} \| \omega^t \|^2 + \frac{2}{\lambda n} s_i (u^*_i - \alpha^*_i) \langle \omega^t, x_i \rangle + \frac{1}{\lambda^2 n^2} s^2_i (u^*_i - \alpha^*_i)^2 \| x_i \|^2
\]
\[
= - s_i (\ell^*_i (-u^*_i) + u^*_i x^T_i \omega^t) + (- \ell^*_i (-\alpha^*_i) - \frac{\lambda n}{2} \| \omega^t \|^2)
\]
\[
+ \frac{s_i}{2} (\gamma_i (1 - s_i) - \frac{1}{\lambda n} s_i \| x_i \|^2) (u^*_i - \alpha^*_i)^2 + s_i (\ell^*_i (-\alpha^*_i) + \alpha^*_i x^T_i \omega^t).
\]
Therefore
\[
A_i - B_i \geq s_i [\ell (x^*_i \omega^t) + \ell^*_i (-\alpha^*_i) + \alpha^*_i x^T_i \omega^t + \frac{1}{2} (\gamma_i (1 - s_i) - \frac{1}{\lambda n} s_i \| x_i \|^2) (u^*_i - \alpha^*_i)^2].
\]
(4.15)
We take the expectation of (4.15), where the expectation is over the samples in the dataset, weighted by \( p_i \). We obtain
\[
\frac{1}{s} \mathbb{E} [A_i - B_i] = \frac{1}{s} \sum_{i=1}^{n} p_i [A_i - B_i] = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{s_i} [A_i - B_i]
\]
\[
= \frac{1}{n} \sum_{i=1}^{n} [\ell (x^*_i \omega^t) + \ell^*_i (-\alpha^*_i) + \alpha^*_i x^T_i \omega^t + \frac{1}{2} (\gamma_i (1 - s_i) - \frac{1}{\lambda n} s_i \| x_i \|^2) (u^*_i - \alpha^*_i)^2].
\]
Note that

\[ f(w^t) - D(\alpha^t) = \frac{1}{n} \sum_{i=1}^n \ell(x_i^t w^t) + \lambda r(w^t) - [\frac{1}{n} \sum_{i=1}^n \ell^*(\alpha^t_i) - \lambda r^*(v(\alpha^t))]. \]

\[ = \frac{1}{n} \sum_{i=1}^n \ell(x_i^t w^t) + \frac{1}{n} \sum_{i=1}^n \ell^*(\alpha^t_i) + \lambda \langle w^t, v(\alpha^t) \rangle \]

\[ = \frac{1}{n} \sum_{i=1}^n [\ell(x_i^t w^t) + \ell_i^*(\alpha^t_i) + \alpha_i^t x_i^t w^t]. \] (4.16)

Here, we denote

\[ G^t = \frac{1}{n} \sum_{i=1}^n [s_i \|x_i\|^2 - \gamma_i (1 - s_i) \lambda n (u_i^t - \alpha_i^t)^2]. \]

Note that in the Lemma 4.9, only the expectation on the left side and the term \( G^t \) are dependent on the probabilities at iteration \( t \).

From Lemma 4.9, to maximize the dual ascent of \( t \)-th update, we should choose \( s \) and \( p \) to be the solution of the following optimization

\[ \max_{s/(p,n) \in [0,1], p \in \mathbb{R}^n, \sum_{i=1}^n p_i = 1} s \mathbb{E}[f(w^t) - D(\alpha^t)] - \frac{s}{n^2} \frac{G^t}{2\lambda}. \]

To solve it, we relax the condition to

\[ \max_{s/(p,n) \in [0, \frac{\lambda n \gamma_i}{1 + \lambda n \gamma_i}], p \in \mathbb{R}^n, \sum_{i=1}^n p_i = 1} s \mathbb{E}[f(w^t) - D(\alpha^t)] - \frac{s}{n^2} \frac{G^t}{2\lambda}. \]

**Lemma 4.10 ([35]) The following optimization problem**

\[ \max_{s \in \mathbb{R}, p \in \mathbb{R}^n} s \text{ s.t. } \frac{s}{p_i n} \in [0, \frac{\lambda n \gamma_i}{1 + \lambda n \gamma_i}], p_i \geq 0, \sum_{i=1}^n p_i = 1 \]

has a solution

\[ s = \frac{n}{n + \sum_{i=1}^n \frac{1}{\lambda n \gamma_i}}, p_i = \frac{1 + \frac{1}{\lambda n \gamma_i}}{n + \sum_{j=1}^n \frac{1}{\lambda n \gamma_j}}. \]

**Proof** We can easily verify the solution holds.

**Remark 4.11** When all \( \gamma_i \) \( (i = \{1, \ldots, n\}) \) are equal, then \( p_i = \frac{1}{n} \). We recover the uniform SDCA here.
4.2. Non-Uniform Stochastic Dual Coordinate Ascent

Here we define \( s_i = \frac{s}{p_in} \). Also, if we assume that \( \ell((w,x_i),y_i) \) is a \( L_i \)-Lipschitz-continuous convex loss function (with respect to the second variable), we have \( \|w_i\| \leq L_i \) and \( \|\alpha_i\| \leq L_i \), yielding

\[
G^t = \frac{1}{n} \sum_{i=1}^{n} (s_i \|x_i\|^2 - \gamma_i (1-s_i) \lambda n) (u_i^t - \alpha_i^t)^2 \leq \frac{1}{n} \sum_{i=1}^{n} (\frac{s}{n p_i} \|x_i\|^2) 4 L_i^2 \leq \frac{4s}{n^2} \sum_{i=1}^{n} \|x_i\|^2 - \frac{4s}{n} L_i^2
\]

To minimize the duality gap, we have another optimization problem

\[
\min \sum_{i=1}^{n} \frac{\|x_i\|^2}{p_i} L_i^2
\]

In our case, \( L_i = 1 \), and the solution is

\[
p_i = \frac{\|x_i\|^2}{\sum_{j=1}^{n} \|x_j\|^2}
\]

which is identical to Non-Uniform SGD.

**Theorem 4.12 ([35])** Consider the SDCA updates given by (4.12), we run NonUnifSDCA (Algorithm 2) by starting from \( w^1 = 0 \) and \( \alpha^1 = 0 \). For any \( \epsilon > 0 \) and \( T \geq (n + \sum_{i=1}^{n} \frac{1}{\lambda_i}) \log((n + \sum_{i=1}^{n} \frac{1}{\lambda_i}) \frac{1}{\epsilon}) \), the following inequality holds with any constant probability \( p \)

\[
E[f(w^{T+1})] - f(w^*) \leq E[f(w^{T+1}) - D(\alpha^{T+1})] \leq \epsilon
\]

**Proof** Denoting \( \epsilon' \) as the difference between \( D(\alpha^t) \) with \( D(\alpha^*) \), where \( \alpha^* \) is the optimal solution of the dual problem. We already have

\[
f(w^t) - D(\alpha^t) \geq D(\alpha^*) - D(\alpha^t) := \epsilon'
\]

We also have \( D(\alpha^{t+1}) - D(\alpha^t) = [D(\alpha^*) - D(\alpha^t)] - [D(\alpha^*) - D(\alpha^{t+1})] = \epsilon' - \epsilon^{t+1} \). Combined with Lemma 4.14 and we take expectation on the sampling probability, we have

\[
E[D(\alpha^{t+1}) - D(\alpha^t)] = E[\epsilon' - \epsilon^{t+1}] \geq \frac{s}{n} E[\epsilon'] - \frac{s}{2 \lambda n^2} G^t
\]

From Lemma 4.9, \( I_i \) is a \((1/\gamma_i)\)-smooth function, so the following inequality holds

\[
(s_i \|x_i\|^2 - \gamma_i (1-s_i) \lambda n) \leq 0
\]

\( \forall t, G^t \leq 0, \)

\[
E[\epsilon^{t+1}] \leq (1 - \frac{s}{n}) E[\epsilon'] \leq (1 - \frac{s}{n})^t E[\epsilon'].
\]
4. Stochastic Optimization with Non-Uniform Sampling

In addition, since
\[
f(0) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(0) + \lambda r(0) \leq 1
\]
and
\[
D(0) = \frac{1}{n} \sum_{i=1}^{n} -\ell_i^*(0) - \lambda r^*(0) = \frac{1}{n} \sum_{i=1}^{n} \max(0 - \ell_i(u_i)) - \max(0 - r(u))
\]
\[
= \frac{1}{n} \sum_{i=1}^{n} \min_{u_i} \ell_i(u_i) + \lambda r(u) \leq 0,
\]
we have \( \epsilon^1 \leq f(0) - D(0) \leq 1 \). Combining with inequality (4.18), we obtain
\[
\mathbb{E}[\epsilon^{t+1}] \leq (1 - \frac{s}{n})^{t} \mathbb{E}[\epsilon^1] \leq \exp(-\frac{st}{n}) \quad (4.19)
\]

By setting
\[
t \geq (n + \frac{n}{\lambda n \gamma_i}) \log\left(\frac{1}{\epsilon}\right),
\]
the algorithm will achieve \( \mathbb{E}[\epsilon^{t+1}] \leq \epsilon \). Furthermore, by Lemma 4.9,
\[
\mathbb{E}[f(w^{t+1}) - D(\alpha^{t+1})] \leq \frac{n}{s} \mathbb{E}[\epsilon^{t+1} - \epsilon^{t+2}] \leq \frac{n}{s} \mathbb{E}[\epsilon^{t+1}].
\]

With
\[
T \geq (n + \sum_{i=1}^{n} \frac{1}{\lambda n \gamma_i}) \log\left(\frac{(n + \sum_{i=1}^{n} \frac{1}{\lambda n \gamma_i}) \frac{1}{\epsilon}}{s}\right),
\]
we obtain \( \mathbb{E}[\epsilon^{T+1}] \leq \frac{s}{n} \epsilon \), and \( \mathbb{E}[f(w^{T+1}) - D(\alpha^{T+1})] \leq \epsilon \).

Here the difference to [35] is that they generalize the definition of \( G \) with a parameter \( R = \sup_{u \neq 0} \|u\|_{D'} / \|u\|_D \), where \( R = 1 \) in our case.
4.2. Non-Uniform Stochastic Dual Coordinate Ascent

4.2.2 Algorithm

The detailed algorithm of NonUnifSDCA is shown as follows. In Algorithm 2, we have two ways to initialize $p$ from Lemma 4.10.

\begin{algorithm}[h]
\caption{NonUnifSDCA (Non-Uniform Stochastic Dual Coordinate Ascent)}
\textbf{Input:} $\lambda > 0$
\textbf{Data:} $\{(x_i, y_i)\}_{i=1}^n$
\textbf{Initialize:} $\alpha^1 = 0$, $w^1 = 0$, probabilities $p_i = \frac{1 + \frac{1}{\lambda n} \gamma_i}{\sum_{j=1}^n \frac{1}{\lambda n} \gamma_j}$ or $p_i = \frac{\|x_i\|}{\sum_{j=1}^n \|x_j\|}$, $\forall i \in \{1, \ldots, n\}$.
\For{$t = 1, 2, \ldots, T$} {
\text{Sample } i_t \text{ from } \{1, \ldots, n\} \text{ based on } p_i;\\
\text{Calculate } \Delta \alpha_{i_t}^t \text{ using following formulas:}\\
\Delta \alpha_{i_t}^t = \arg \max_{\Delta \alpha_{i_t}^t} \left[ -\frac{1}{\lambda n} \|w^t + \frac{1}{\lambda n} \Delta \alpha_{i_t}^t x_{i_t}\|^2 - \ell_{i_t}(-(\alpha_{i_t}^t + \Delta \alpha_{i_t}^t)) \right];\\
\text{Set } \alpha_{i_t}^{t+1} \leftarrow \alpha_{i_t}^t + \Delta \alpha_{i_t}^t;\\
\text{Set } w^{t+1} \leftarrow w^t + \frac{1}{\lambda n} \Delta \alpha_{i_t}^t x_{i_t};
}
\textbf{Output:} $w^{T+1}$
\end{algorithm}
In this chapter, we investigate adaptive non-uniform sampling. We will introduce AdaSGD first, followed by its variance reduction variant AdaSVRG. Then we will introduce AdaSDCA and AdaSDCAS, a variant of SDCA for subgradients. Finally, we will compare our algorithms with two existing ones, AdaGrad and Csiba-AdaSDCA+. For SGD and SDCA, we will first explain the ideas behind it and then give the detailed algorithms and proofs about convergence rate.

5.1 Adaptive Stochastic Gradient Descent

5.1.1 Idea behind SGD

In this Subsection, we will derive the probabilities from the gap of expected values of objective function.

By assuming $\ell$ is a $1/\gamma$-smooth function and $\eta_t \leq \gamma$, we have

$$
\ell(w^{t+1}) = \ell(w^t) - \eta_t \nabla \ell(w^t)^\top g_i^t + \frac{\eta_t^2}{2\gamma} \|g_i^t\|^2 
$$

$$
\leq \ell(w^t) - \eta_t \nabla \ell(w^t)^\top g_i^t + \frac{\eta_t}{2} \|g_i^t\|^2.
$$

Suppose the probability distribution stays the same for iteration $t$ and $t + 1$. Consequently, the expectations over $f(w^{t+1})$ and $f(w^t)$ are the same. So we have

$$
\mathbb{E}[f(w^{t+1})] - \mathbb{E}[f(w^t)] = \mathbb{E}[\ell(w^{t+1}) + \lambda r(w^{t+1})] - \mathbb{E}[\ell(w^t) + \lambda r(w^t)]
$$
Towards Adaptive Non-Uniform Sampling

\[
\begin{align*}
&\leq -\mathbb{E}[\eta_t \nabla \ell(w^t)^\top g^t_i] + \mathbb{E}\left[\frac{\eta_t}{2} \|g^t_i\|^2\right] + \mathbb{E}\left[\lambda \frac{\|w^{t+1}\|^2 - \|w^t\|^2}{2}\right] \\
&= -\mathbb{E}[\eta_t \nabla \ell(w^t)^\top g^t_i] + \mathbb{E}\left[\frac{\eta_t}{2} \|g^t_i\|^2\right] + \mathbb{E}\left[\lambda \frac{\|w^t + \eta_t g^t_i\|^2 - \|w^t\|^2}{2}\right] \\
&= \mathbb{E}\left[\frac{\eta_t}{2} \|g^t_i\|^2\right] + \frac{\lambda \eta_t^2}{2} \mathbb{E}[\|g^t_i\|^2] - \eta_t \langle \chi^t_i, \nabla \ell(w^t) \rangle + \lambda \eta_t \langle w^t, \chi^t_i \rangle \\
&= \frac{\eta_t}{2} (1 + \lambda \eta_t) \mathbb{E}[\|g^t_i\|^2] - \eta_t \langle \chi^t_i, \nabla \ell(w^t) \rangle + \lambda \eta_t \langle w^t, \chi^t_i \rangle.
\end{align*}
\]

According to the above formula, we can reduce the objective function by solving the following optimization problem:

\[\min \mathbb{E}[\|g^t_i\|^2].\]

By the Cauchy-Schwarz inequality and the fact that \(\sum_{i=1}^n p_i = 1\),

\[\mathbb{E}[\|g^t_i\|^2] = \sum_{i=1}^n \frac{\|\chi_i\|^2}{n^2 p_i} = \left(\sum_{i=1}^n \frac{\|\chi_i\|^2}{n^2 p_i}\right) \left(\sum_{i=1}^n p_i\right) \geq \left(\sum_{i=1}^n \frac{\|\chi_i\|}{n}\right)^2.
\]

The above inequality holds when

\[p_i = \frac{\|\chi_i\|}{\sum_{j=1}^n \|\chi_j\|}.\quad (5.1)
\]

5.1.2 Adaptive Sampling for SGD

Based on the above result, we suggest modifying the NonUnifSGD (Algorithm 1) with adaptive sampling. The adaptive sampling is incorporated into the Non-Uniform SGD algorithm as follows: in each epoch (epoch=n updates), we record \(c_i\), the maximum value of \(\chi_i(w)\) of the last \(k \in \mathbb{N}_+\) iterations of the current epoch. At the start of the next epoch, we update \(p_i\) according to \(c_i\).

We discuss the impact of different values of \(k\) on the complexity in Chapter 7 and compare the performance of different values of \(k\) in Chapter 8.

Two Schemes

Notice that for any iteration \(t \in \{1, \ldots, T\}\), if there is a \(i \in \{1, \ldots, n\}\) with \(\chi_i\) non-zero but its corresponding \(p_i \to 0\). From (4.4), we have \(g^t_i = \frac{\chi_i}{p_i} \to \infty\) and consequently

\[\mathbb{E}[\|g^t_i\|^2] \to \infty.\quad (5.2)
\]

Therefore, it is essential to set each \(p_i\) larger than some minimum value in practice. We implement two schemes for adaptive SGD, which are named as
‘Aggressive’ and ‘Conservative’. Before the end of each epoch (i.e. performing $n$ stochastic updates, where $n$ is the number of datapoints), both schemes perform $k$ additional passes through the data to update the probabilities. These $k$ passes calculate $c_i = \max \| \chi_i \|$ for each sample $i \in \{1, \ldots, n\}$.

For the ‘Aggressive’ model, we update the sampling probability as follows:

- We set the probability of $i$ proportional to the recorded value $c_i$.

For the ‘Conservative’ model,

- If a sample is correctly classified during all the last $k$ iterations, we set its probability to some small value;
- Otherwise, we set the probability of $i$ proportional to the recorded value $c_i$.

In the Aggressive model, the weight of the correctly classified samples $c_i = \lambda \| w \|$ while that is 1 in Conservative model. Assume $\lambda \| w \| \ll 1$, Conservative model increases the weight of correctly classified samples from $\lambda \| w \|$ to 1. These two updating algorithms are useful in practice because both of them decrease the probability of correctly classified points. We will compare the impact on performance in Chapter 8.

Definition 5.1 $\mathbb{1}_i$ is an indicator function which returns 1 if point $i$ is correctly classified during all the $k$ iterations, otherwise returns 0.

See Algorithm 3 and Algorithm 4 for details. We consider the points as correctly classified if $\mathbb{1}_i = 1$. The difference between the two schemes is that Conservative model increases the weights of the correctly classified points more.

Pseudo Code for Updating the Probability

From the code we can see that Algorithm 3 updates the probability $p_i$ proportional to $c_i$.

**Algorithm 3: Aggressive Probability Update**

```plaintext
for $j = 1, \ldots, n$
    set $p_j \leftarrow \frac{c_j}{\sum_{k=1}^{n} c_k}$
end
```

In Algorithm 4, the probabilities of correctly classified points are updated
5. Towards Adaptive Non-Uniform Sampling

proportional to 1 while other points are updated proportional to \( c_i \).

**Algorithm 4: Conservative Probability Update**

Set \( s \leftarrow \sum_{j=1,\ldots,n} c_j \);  
Set \( c \leftarrow |S| \) where \( S \leftarrow \{j | \mathbb{I}_j = 1\} \);  
for \( j = 1,\ldots,n \)  
\[ p_j > 0 ? \quad p_j \leftarrow \frac{c_j}{s+c} : \quad p_j \leftarrow \frac{1}{s+c}; \]
end

**Multi-Stage Adaptive Algorithm**

The detailed algorithm is shown in AdaSGD (Algorithm 5). The differences between NonUnifSGD (Algorithm 1) and AdaSGD (Algorithm 5) are

1. At the beginning of each epoch, we update the probability.
2. In the last \( k \) iterations of each epoch, we do a pass of the whole dataset to calculate the subgradient for each point.
3. At the end of each epoch, we keep track of the maximum subgradient of each point in the last \( k \) iterations.

In order to analyze the convergence rate of adaptive SGD, we distinguish between the two expectations:

1. \( \mathbb{E}_t \): probabilities AT iteration \( t \);  
2. \( \mathbb{E} \): randomness of all steps UNTIL iteration \( t \) (included).

**Theorem 5.2** Suppose \( f \) is a \( \lambda \)-strongly convex function. If we choose the stepsize as \( \eta_t = \frac{1}{\lambda t} \), then after \( T \) iterations of AdaSGD (Algorithm 5) with starting point \( w^1 = 0 \), it holds that

\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{t=1}^T w^t \right) \right] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^T \mathbb{E} \left[ \|g^t_i\|^2 \right] \frac{1}{t}
\]

where \( g^t_i = \frac{\chi_t^i(w^t)}{np^t_i} \) and \( \mathbb{E} \) is the expectation taken with respect to the whole procedure of the algorithm where \( p^t \) is the probability at time \( t \).

**Proof** We again use Inequality (4.5), which is about the improvement conditioned on the last step for non-uniform probabilities. We choose \( \eta = \frac{1}{\lambda t} \), giving us:

\[
f(w^t) - f(w^*) \leq \frac{1}{2\lambda t} \mathbb{E}_t[\|g^t_i\|^2 \|w^t\|] + \frac{\lambda(t-1)}{2} \|w^t - w^*\|^2 - \frac{\lambda t}{2} \mathbb{E}_t\|w^{t+1} - w^*\| \|w^t\|^2
\]
5.1. Adaptive Stochastic Gradient Descent

Note that we put a \( t \) as a subscript of the expectation \( \mathbb{E} \) to denote the changing probabilities on the fly, because the probability at iteration \( t \) is not necessarily to that at iteration \( t-1 \).

Taking the expectation over the randomness of the whole algorithm, which denoted as \( \mathbb{E} \), we get

\[
\mathbb{E}[f(w^t)] - f(w^*) \leq \frac{1}{2\lambda t} \mathbb{E}[\|g^t_i\|^2] + \frac{\lambda(t-1)}{2} \mathbb{E}[\|w^t - w^*\|^2] - \frac{\lambda t}{2} \mathbb{E}[\|w^{t+1} - w^*\|^2]
\]

Summing up from iteration 1 to iteration \( T \), we get

\[
\sum_{t=1}^{T} \mathbb{E}[f(w^t)] - Tf(w^*) \leq \sum_{t=1}^{T} \frac{1}{2\lambda t} \mathbb{E}[\|g^t_i\|^2] + \sum_{t=1}^{T} \frac{\lambda(t-1)}{2} \mathbb{E}[\|w^t - w^*\|^2]
\]

\[
- \sum_{t=1}^{T} \frac{\lambda t}{2} \mathbb{E}[\|w^{t+1} - w^*\|^2] = \sum_{t=1}^{T} \frac{1}{2\lambda t} \mathbb{E}[\|g^t_i\|^2] - \mathbb{E}[\|w^{T+1} - w^*\|^2]
\]

We use convexity of the function \( f \) again by Jensen’s inequality:

\[
\mathbb{E}\left[ f\left( \frac{1}{T} \sum_{t=1}^{T} w^t \right) \right] - f(w^*) \overset{\text{Jensen}}{\leq} \mathbb{E}\left[ \frac{1}{T} \sum_{t=1}^{T} f(w^t) \right] - f(w^*)
\]

\[
= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[f(w^t)] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g^t_i\|^2]}{t} - \frac{\lambda}{2} \mathbb{E}[\|w^{T+1} - w^*\|^2]
\]

\[
\leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g^t_i\|^2]}{t}
\]

\( \square \)

Note that Theorem 5.2 is identical to Theorem 4.1. Here we prove that although probability distribution in different iterations varies in adaptive algorithms, the basic convergence theorem holds as well. Similar to Corollary 4.3 and 4.4, we can also get simple convergence rates by bounding the subgradient. We have corollaries for adaptive algorithms as following.

**Corollary 5.3** Suppose \( f \) is a \( \lambda \)-strongly convex function, and assume that according to Definition 4.2, \( \max \{ \|x^t_i(w^t)\|^2 \} \leq G \). If we choose \( \eta_t = \frac{1}{\lambda t} \) and assume \( p_i > \varepsilon \) for all \( i = \{1, \ldots, n\} \), then from Theorem 5.2,

\[
\mathbb{E}\left[ f\left( \frac{1}{T} \sum_{t=1}^{T} w^t \right) \right] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{\mathbb{E}[\|g^t_i\|^2]}{t} \leq \frac{1}{2\lambda T} \sum_{t=1}^{T} \frac{G}{\varepsilon n t} \leq \frac{G(\ln T + 1)}{2\lambda \varepsilon n T}
\]

35
Corollary 5.4 Suppose \( f \) is a \( \lambda \)-strongly convex function, and assume that according to Definition 4.2, \( \mathbb{E} [ \| \chi_i^t (w^t) \|^2 ] \leq W \) for all \( t \). If we choose \( \eta_t = \frac{1}{\lambda t} \) and assume \( \eta_t > \epsilon \) for all \( i = \{1, \ldots, n\} \), then from Theorem 5.2,

\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{i=1}^{T} w^i \right) \right] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{i=1}^{T} \mathbb{E} [\|g_i^t\|^2] - \frac{1}{2\lambda T} \sum_{i=1}^{T} \frac{W}{n^2\epsilon^2 t} \leq \frac{W(\ln T + 1)}{2\lambda T n^2 \epsilon^2}
\]

Definition 5.5 Define \( B_i^* = \| \chi_i(w^*) \| \), let \( B = \max\{B_i^* | i \in \{1, \ldots, n\}\} \).

Theorem 5.6 Suppose \( f \) is a \( \lambda \)-strongly convex function, we run AdaSGD (Algorithm 5) and assume that \( w \) starts with \( w^1 = 0 \) and converges to \( w^* \). We use the continuity of the loss function \( \ell \) and subgradient \( \chi \) of \( w \), and the fact that the convergence of \( \chi_i (w) \) to \( \chi_i (w^*) \) for all \( i \). We further assume \( \eta_t > \epsilon \) for all \( i = \{1, \ldots, n\} \) and \( \| \chi_i (w^t) \| - \| \chi_i (w^*) \| \leq \frac{B_i^*}{\lambda} \), such that we have \( \| \chi_i (w^t) \| \leq \| \chi_i (w^*) \| + \frac{B_i^*}{\lambda} \leq (1 + \frac{1}{\lambda}) B_i^* \). Then the following inequality holds,

\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{i=1}^{T} w^i \right) \right] - f(w^*) \leq \frac{B^2}{2\lambda \epsilon n T} (\ln T + 1 + 2 H_{T}^{(2)} + H_{T}^{(3)})
\]

where the expectation is taken with respect to the distribution \( p^t \) and \( H_{T}^{(a)} = \sum_{i=1}^{T} \frac{1}{i^a} \) is \( T \)-th harmonic number with order \( a \). The series of harmonic numbers are bounded by a small constant.

Proof

\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{i=1}^{T} w^i \right) \right] - f(w^*) \leq \frac{1}{2\lambda T} \sum_{i=1}^{T} \mathbb{E} [\|g_i^t\|^2] - \frac{1}{2\lambda T} \sum_{i=1}^{T} \frac{(1 + \frac{1}{\lambda})^2 \max\{B_i^*\}^2}{\epsilon n T} \leq \frac{B^2}{2\lambda \epsilon n T} (\ln T + 1 + 2 H_{T}^{(2)} + H_{T}^{(3)})
\]

\( B \) is determined by the maximum subgradient of all the samples calculated with \( w^* \) and \( G \) is determined by the upper bound of subgradient calculated with all other \( w \). Trivially, \( G > B^2 \). Compared to Corollary 4.3, we reduce the number of iterations by approximately a factor of \( \frac{G}{\lambda} \).

The new algorithm does the following: in the first stage, it runs AdaSGD (Algorithm 5) with fixed \( p \) for one epoch. Then we update the probability according to the subgradient bounds for some \( t \), where it holds that \( \| \chi_i^t \| \leq c_i \) for each \( i \). \( c_i \) is the maximum subgradient we obtain in most recent \( k \) steps.

We can use either of the above mentioned two schemes to update the probabilities — Aggressive or Conservative.

In Chapter 8, we show the results of AdaSGD and discuss what value of \( k \) we should use.
5.1. Adaptive Stochastic Gradient Descent

**Algorithm 5: AdaSGD (Adaptive Non-Uniform Stochastic Gradient Descent)**

**Input:** \( \lambda > 0 \)

**Data:** \( \{(x_i, y_i)\}_{i=1}^n \)

**Initialize:** \( w^0 = 0 \), probabilities
\[
p_i = \frac{\|x_i\|^2 + \sqrt{\lambda}}{\sum_{j=1}^n \|x_j\|^2 + \sqrt{\lambda}}, c_i = 0, \forall i \in \{1, \ldots, n\}.
\]

**for** \( t = 1, 2, \ldots, T \)

Sample \( i_t \) from \( \{1, \ldots, n\} \) based on \( p \);

Set \( \eta_t \leftarrow \frac{1}{\sqrt{\lambda}} \);

Calculate \( \ell' \leftarrow \ell'(\langle x_{i_t}, w^t \rangle, y_{i_t}) \);

Set \( \chi_{i_t}^t(w^t) \leftarrow \ell' x_{i_t} + \lambda \nabla r(w^t) \);

if \( (t-1) \mod n \geq n-k \) then

**for** \( i = 1, 2, \ldots, n \)

Calculate \( \ell'(\langle x_i, w^t \rangle, y_i) \);

Set \( \chi_i \leftarrow \ell'(\langle x_i, w^t \rangle, y_i)x_i + \lambda \nabla r(w^t) \);

Set \( c_i \leftarrow \max\{c_i, \|\chi_i\|\} \);

end

if \( t \mod n = 0 \) then

**Option I:** Run Algorithm 3 (Aggressive Update);

**Option II:** Run Algorithm 4 (Conservative Update);

end

Set \( g_{i_t}^t \leftarrow \frac{\chi_{i_t}^t(w^t)}{n p_{i_t}} \);

Set \( w^{t+1} \leftarrow w^t - \eta_t g_{i_t}^t \);

end

**Output:** \( w^{T+1} \)

**Theorem 5.7** Suppose \( f \) is a \( \lambda \)-strongly convex function. If we choose the stepsize to be \( \eta_t = \frac{1}{\lambda} \), then after running AdaSGD (Algorithm 5) for \( E \in \mathbb{N}_+ \) epochs with starting point \( w^1 = 0 \) and assume \( \sum_t \frac{\|\chi_i(w^t)\|^2}{t} \leq \frac{(B^*)^2}{(2e-1)(2e+1)} \), for \( (e-1)n < t \leq en \), it holds that
\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{t=1}^T w^t \right) \right] - f(w^*) \leq \frac{3B^2}{8\lambda en T}
\]

where \( T = nE \).

**Proof**
\[
\mathbb{E} \left[ f \left( \frac{1}{T} \sum_{t=1}^T w^t \right) \right] - f(w^*) \leq \frac{1}{2\lambda en} \sum_{t=1}^{nE} \frac{\mathbb{E} \|g_{i_t}^t\|^2}{t}
\]
5. Towards Adaptive Non-Uniform Sampling

\[ \leq \frac{1}{2\lambda n} E \sum_{t=1}^{E} B^2 2ne \left( \frac{1}{3} + \frac{1}{2} - \frac{1}{4} + \ldots + \frac{1}{2E - 1} - \frac{1}{2E + 1} \right) < \frac{3B^2}{8\lambda nT} \]

Compared with Theorem 5.6, we further reduce the convergence rate by approximately a factor of \( \frac{1}{4} \).

5.2 Adaptive Stochastic Variance Reduced Gradient Descent

In this section, we derive AdaSVRG (a variant of AdaSGD) which is inspired by [32].

5.2.1 Idea behind SVRG

We add a \( \tilde{w} \) (which denotes the \( w \) of last epoch) for a new update equation. Therefore, we get

\[ w^{t+1} := w^t - \eta_t [g^t_i(w^t) - g^t_i(\tilde{w}) - \nabla f(\tilde{w})] \]  \hspace{1cm} (5.3)

The expectation of the update function is still the same as before, because

\[ E[g(w) - g(\tilde{w}) + \nabla f(\tilde{w})] = E[g(w)] - E[g(\tilde{w})] + \nabla f(\tilde{w}) = \nabla f(w). \]

Inspired by this idea, we derive a new adaptive algorithm which we call AdaSVRG (Adaptive Stochastic Variance Reduced Gradient) by changing the update equation of \( w^{t+1} \). In contrast to the constant stepsize in [32], we still use \( \eta = \frac{1}{\lambda t} \) in AdaSVRG (Algorithm 6). In Subsection 8.3.6, we compare the performance of AdaSVRG with other adaptive algorithms.

5.3 Adaptive Stochastic Dual Coordinate Ascent

5.3.1 Idea behind SDCA

For SDCA, we could use both duality gap and subgradient \( \chi \) for updating the probability. The idea of using the duality gap comes naturally as the \( n \) individual gaps always sum up to the total gap (by the definition of Empirical Risk Minimization). The total duality gap is a natural measure of the current approximation quality. The subgradient \( \chi \) is inspired by Subsection 5.1.1.

The detailed explanation is given in the following subsections. We compare the experimental performance of both schemes in Subsection 8.3.5.
Algorithm 6: AdaSVRG (Adaptive Stochastic Variance Reduced Gradient Descent)

**Input:** $\lambda > 0$

**Data:** $\{(x_i, y_i)\}_{i=1}^n$

**Initialize:** $\alpha = 0$, $\omega^0 = \tilde{\omega} = 0$, probabilities

$$p_i = \frac{\|x_i\|^2 + \sqrt{T}}{\sum_{j=1}^n \|x_i\|^2 + \sqrt{T}}, c_i = 0, \forall i \in \{1, \ldots, n\}$

for $t = 1, 2, \ldots, T$

Sample $i_t$ from $\{1, \ldots, n\}$ based on $p$;

Set $\eta_t \leftarrow \frac{1}{\lambda t}$;

Set $\chi_{i_t}^t(\omega^t) \leftarrow \ell'(\langle x_{i_t}, \omega^t \rangle, y_i)x_{i_t} + \lambda \nabla r(\omega^t)$;

Set $\chi_{i_t}^t(\tilde{\omega}) \leftarrow \ell'(\langle x_{i_t}, \tilde{\omega} \rangle, y_i)x_{i_t} + \lambda \nabla r(\tilde{\omega})$;

if $(t-1) \mod n \geq n-k$ then

    for $i = 1, 2, \ldots, n$

        Calculate $\ell'(\langle x_i, \omega^t \rangle, y_i)$;

        Set $\chi_i \leftarrow \ell'(\langle x_i, \omega^t \rangle, y_i)x_i + \lambda \nabla r(\omega^t)$;

        Record $c_i \leftarrow \max\{c_i, \|\chi_i\|\}$;

end

if $t \mod n = 0$ then

    **Option I:** Run Algorithm 3 (Aggressive Update);

    **Option II:** Run Algorithm 4 (Conservative Update);

    $\tilde{\omega} = \omega^t$;

end

Set $g_{i_t}^t(\omega^t) \leftarrow \frac{\chi_{i_t}^t(\omega^t)}{\eta_t}$;

Set $g_{i_t}^t(\tilde{\omega}) \leftarrow \frac{\chi_{i_t}^t(\tilde{\omega})}{\eta_t}$;

Set $\omega^{t+1} \leftarrow \omega^t - \eta_t[g_{i_t}^t(\omega) - g_{i_t}^t(\tilde{\omega}) + \nabla f(\tilde{\omega})]$;

end

**Output:** $\omega^{T+1}$
5. Towards Adaptive Non-Uniform Sampling

5.3.2 Using Duality Gaps to Update the Probabilities

**Definition 5.8** Define the gap of point $i$ as

$$
\sigma_i^t = \ell(x_i^Tw^t) + \ell^*(-a_i^t) + a_i^tx_i^Tw^t
$$

where $w^t$ here is assumed to be the corresponding primal vector for the current $\alpha^t$, that is $w^t(\alpha^t) := \frac{1}{n}\sum_{i=1}^{n} a_ix_i^t$.

From (4.16), the duality gap between the primal objective and dual objective at the $t$-th iteration is defined as

$$
f(w^t) - D(\alpha^t) = \frac{1}{n}\sum_{i=1}^{n} \sigma_i^t.
$$

Note that, duality gap is always greater than or equal to 0. When it is zero, we consider the function as holds with strong duality.

We are going to use the same adaptive sampling strategy as described in Chapter 5 for SGD, which records the recent $k$ gaps for each point before an epoch ends, and update all $p_i = \frac{c_i}{\sum_{j=1}^{k} c_j}$ correspondingly. Here $c_i$ is the maximum gap among all the $k$ gaps. $c_i := \max_t \sigma_i^t$ and $t$ belongs to these $k$ iterations.

See Algorithm 7 for details. Note that, we only use Aggressive Update (Algorithm 3) for this method.

5.3.3 Using Subgradients to Update the Probabilities

We can adopt the ideas in Subsection 5.1.1 for SDCA to derive AdaSDCAS (short for AdaSDCA by Subgradient). The difference between 7 and 8 is the calculation of $c_i$. In Algorithm 7, $c_i$ is calculated by the gap $\sigma_i$ while in Algorithm 8, $c_i$ is calculated by the subgradient $\chi_i$.

**Theorem 5.9** Consider the SDCA updates given by (4.12), we run AdaSDCA (Algorithm 7) of $E$ epochs by starting from $\alpha^1 = 0$ and $w^1 = 0$, here $T = nE$.

Assume $\forall (e-1)n \leq t \leq en$, $\mathbb{E}(|e^t|) \leq \left[1 - \frac{1}{n}\right] \mathbb{E}(|e^{t-1}|) \exp(\frac{c^t}{n})$ where $e^t$ is the error at time $t$. For any $\epsilon > 0$ and

$$
T \geq \frac{1}{n} + \frac{1}{n + \sum_{i=1}^{n} \frac{1}{\lambda n \gamma_i}} \log\left(\frac{n + \sum_{i=1}^{n} \frac{1}{\lambda n \gamma_i}}{\epsilon}\right),
$$

the following inequality holds with probability $p^T$, over which the expectation is taken

$$
\mathbb{E}[f(w^{T+1}) - f(w^*)] \leq \mathbb{E}[f(w^{T+1}) - D(\alpha^{T+1})] \leq \epsilon
$$
5.3. Adaptive Stochastic Dual Coordinate Ascent

Algorithm 7: AdaSDCA (Adaptive Non-uniform Stochastic Dual Coordinate Ascent)

Input: $\lambda > 0$

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

Initialize: $\alpha^1 = 0$, $w^1 = 0$, probabilities $p_i = \frac{1 + \frac{1}{n^2} \lambda}{\sum_{j=1}^n |x_j|^2}$ or $p_i = \frac{\|x_i\|}{\sum_{j=1}^n \|x_j\|}$, $c_i = 0$, $\forall i \in \{1, \ldots, n\}$.

for $t = 1, 2, \ldots, T$

Sample $i_t$ from $\{1, \ldots, n\}$ based on $p_i$;

Calculate $\Delta \alpha_{i_t}^t$ using following formulas:

$\Delta \alpha_{i_t}^t = \arg \max \Delta \alpha_{i_t}^t \left[ \frac{n}{2} \|w^t + \frac{1}{\lambda} \Delta \alpha_{i_t}^t x_{i_t}\|^2 - \ell_{i_t}^* \left( -\alpha_{i_t}^t + \Delta \alpha_{i_t}^t \right) \right]$;

Set $\alpha_{i_t}^{t+1} \leftarrow \alpha_{i_t}^t + \Delta \alpha_{i_t}^t$;

Set $w^{t+1} \leftarrow w^t + \frac{1}{\lambda} \Delta \alpha_{i_t}^t x_{i_t}$;

if $(t-1) \mod n \geq n-k$ then

for $i = 1, 2, \ldots, n$

Calculate $c_i^t \leftarrow \ell(x_i^t \cdot w^t) + l^* \left( -\alpha_i^t \right) + \alpha_i^t \langle x_i, w^t \rangle$;

Set $c_i \leftarrow \max\{c_i, c_i^t\}$;

end

end

if $t \mod n = 0$ then

Option I: Run Algorithm 3 (Aggressive Update);

Option II: Run Algorithm 4 (Conservative Update);

end

Output: $w^{T+1}$

Proof Combine the assumption and (4.19), for any $t$ we get

$$E[\epsilon^{t+1}] \leq (1 - \frac{s}{n})E[\epsilon^t] \exp(-\frac{1}{n})$$

$$\leq [1 - \frac{s}{n}]^t E[\epsilon^1] \exp(-t)$$

$$\leq \exp(-\frac{t}{n + \sum_{j=1}^n \frac{1}{\lambda |x_j|^2}}) \exp(-\frac{t}{n}) \leq \epsilon$$

Hence, for $t \geq \frac{1}{\frac{1}{n} + \sum_{j=1}^n \frac{1}{\lambda |x_j|^2}} \log \frac{1}{\epsilon}$, $E[\epsilon^{t+1}] \leq \epsilon$.

By Lemma 4.9 and Theorem 4.12,
Algorithm 8: AdaSDCAS (Adaptive Non-uniform Stochastic Dual Coordinate Ascent by Subgradient)

**Input:** $\lambda > 0$
**Data:** $\{(x_i, y_i)\}_{i=1}^n$
**Initialize:** $\alpha^1 = 0$, $w^1 = 0$, probabilities $p_i = \frac{1 + \frac{1}{2} n_i}{\sum_{j=1}^n \gamma_j}$ or $p_i = \frac{\|x_i\|}{\sum_{j=1}^n \|x_j\|}$, $c_i = 0$, $\forall i \in \{1, \ldots, n\}$.

for $t = 1, 2, \ldots, T$
  Sample $i_t$ from $\{1, \ldots, n\}$ based on $p$;
  Calculate $\Delta \alpha^{t}_{i_t}$ using following formulas:
  $\Delta \alpha^{t}_{i_t} = \arg \max_{\Delta \alpha^{t}_{i_t}} \left[-\frac{1}{n} \|w^t + \frac{1}{\lambda n} \Delta \alpha^{t}_{i_t} x_{i_t}\|^2 - \ell^{t}_{i_t}(-(\alpha^{t}_{i_t} + \Delta \alpha^{t}_{i_t}))\right]$;
  Set $\alpha^{t+1}_{i_t} \leftarrow \alpha^{t}_{i_t} + \Delta \alpha^{t}_{i_t}$;
  Set $w^{t+1} \leftarrow w^t + \frac{1}{\lambda n} \Delta \alpha^{t}_{i_t} x_{i_t}$;
  if $(t-1) \mod n \geq n-k$ then
    for $i = 1, 2, \ldots, n$
      Calculate $\ell^{t}(\langle x_i, w^{t}\rangle, y_i)$;
      Set $\chi^{t}_{i} \leftarrow \ell^{t}(\langle x_i, w^{t}\rangle, y_i) x_i + \lambda \nabla r(w^{t})$;
      Record $c_i \leftarrow \max\{c_i, \|\chi^{t}_{i}\|\}$;
  end
  if $t \mod n = 0$ then
    Option I: Run Algorithm 3 (Aggressive Update);
    Option II: Run Algorithm 4 (Conservative Update);
  end
end

**Output:** $w^{T+1}$

$\mathbb{E}[f(w^{t+1}) - D(a^{t+1})] \leq \frac{n}{2} \mathbb{E}[\epsilon^{t+1}]$. The inequality holds with

$$T \geq \frac{1}{n} \log((n + \sum_{i=1}^n \frac{1}{\lambda n_i}) \frac{1}{\epsilon}).$$

The difference between Theorem 4.12 and Theorem 5.9 is an assumption on the duality gap, with an additional quotient $\exp(-\frac{1}{n})$. Here the number of iterations have been reduced by a factor of $\frac{2 + \frac{1}{n} \sum_{i=1}^n \frac{1}{\lambda n_i}}{n}$.  

42
5.4 Comparison with Existing Adaptive Algorithms

5.4.1 AdaGrad

AdaGrad [4] changes (4.3) to adaptive update based on historical information. The strategy is that infrequent features are assigned higher learning rates than frequent ones.

\[
\eta_{t,i} = \frac{\eta}{\sqrt{D_{t,ii}}}
\]

Here each \( D_t \in \mathbb{R}^{d \times d} \) is a diagonal matrix where diagonal element \( D_{i,i} \) is defined to be the sum of the squares of the \( i \)th dimension of all previous gradients: \( \sum_{t'=1}^{t} \Delta_{t',i}^2 \).

The AdaGrad update per feature is

\[
w_{t+1} = w_t - \eta D_t^{-1/2} g_t.
\]

The difference between AdaGrad and plain SGD is that AdaGrad sets the value of \( \eta \) to 1 [4] by experimentation, because in this way the learning rate is quickly adapted. Although \( \eta \) is not dependent on \( \lambda \) anymore, the value of \( \lambda \) is still important to the performance.

5.4.2 AdaSDCA+

Csiba et al. [3] independently came up with an adaptive SDCA algorithm, which is coincidentally named AdaSDCA, and the heuristic version of it called AdaSDCA+. For clarity, we shall refer to their algorithms as Csiba-AdaSDCA and Csiba-AdaSDCA+.

Csiba-AdaSDCA has some defects compared to our algorithms as it computes the residue of each sample and then update the corresponding probability during each iteration. Each update obviously requires a pass through the entire dataset. This increases the time complexity by a factor of \( n \), i.e. \( O(n^2) \), where \( n \) is the number of samples of the dataset. The cost of this algorithm is \( n \) times larger than that of NonUnifSDCA.

They also designed a heuristic variant called AdaSDCA+ by changing the probability over the dual residue per epoch and the probability of the sampled point by dividing it by a constant \( m \). But they do not provide any theoretical basis for choosing \( m \), nor do they analyze the convergence rate.

Although both AdaSDCA in this thesis and Csiba-AdaSDCA+ are adaptive algorithms with similar strategies, they are simultaneous and independent work. The main differences between Csiba-AdaSDCA+ and our AdaSDCA are as follows:
Towards Adaptive Non-Uniform Sampling

- Csiba-AdaSDCA+ updates the probability of the sampled point on the fly by a small constant (>1). It also adjusts the probability at the end of one epoch to be proportional to the residue.

- We keep track of the duality gap in the last $k$ iterations of one epoch and update the probability by setting it proportional to the recorded gap of each point. When $k$ is equal to 1, the two algorithms have the same computational cost.

We compare the performance of the two algorithms in greater depth in Subsection 8.3.3.
From Chapter 5, both AdaSGD and AdaSDCA need to pass through the whole dataset to compute $c_i$ (maximum subgradient or maximum gap) for each point, which may be intolerable for some applications. In this chapter, we introduce the online variants of adaptive SGD and SDCA to solve this problem. The differences between online variants and the offline adaptive algorithms are the following:

1. Online variants do not require an additional pass over the dataset to update the probability;
2. Online variants update the probability on-the-fly;
3. Online variants have a minimum probability $\epsilon$, and we never set the probability smaller than this minimum probability.

**Definition 6.1** Define $\epsilon$ as the minimum probability. In online algorithms, $p_i \geq \epsilon$ (for $i = 1, \ldots, n$).

To clarify ourselves, the value of $\epsilon$ is not so important. We use it to avoid extreme large value for $\|g\|$ as $g_i = \frac{x_i}{np_i}$.

### 6.1 Online Adaptive Stochastic Gradient Descent

We modify the AdaSGD to an online adaptive algorithm called Online Adaptive SGD (OASGD), in which we keep all the probabilities $\geq \epsilon$ in order to avoid extreme cases where numerous wrongly classified samples are allocated with small probabilities. After this procedure, we normalize all the $p_i$ to the range $[0, 1]$. 
6. Online Adaptive Non-Uniform Sampling

See Algorithm 9 for details.

**Algorithm 9: OASGD (Online Adaptive Stochastic Gradient Descent)**

**Input:** $\lambda > 0$

**Data:** $\{(x_i, y_i)\}_{i=1}^n$

**Initialize:** $w^1 = 0$, $p_i = \frac{\|x_i\|^2 + \sqrt{\lambda}}{\sum_{j=1}^n \|x_j\|^2 + \sqrt{\lambda}}$, $c_i = 0$, $\forall i \in \{1, \ldots, n\}$.

for $t = 1, 2, \ldots, T$

- Sample $i_t$ from $\{1, \ldots, n\}$ based on $p$;
- Set $\eta_t = \frac{1}{\lambda t}$;
- Calculate $\ell'(\langle x_{i_t}, w^t \rangle, y_{i_t})$;
- Set $\chi'_{i_t}(w^t) \leftarrow \ell'x_{i_t} + \lambda \nabla r(w^t)$;
- Set $g^t_{i_t} \leftarrow \frac{\chi'_{i_t}(w^t)}{np_{i_t}}$;
- Set $w^{t+1} \leftarrow w^t - \eta_t g^t_{i_t}$;
- Set $p_{i_t} \propto \max\{\varepsilon, \|\chi'_{i_t}(w^t)\|\}$;

end

**Output:** $w^{T+1}$

6.2 Online Adaptive Stochastic Dual Coordinate Ascent

Online Adaptive SDCA (OASDCA) is the online variant for AdaSDCA by updating the probability of the sampled point on the fly. It calculates the gap $\sigma_{i_t}$ for that point, and then updates the probability according to $\sigma_{i_t}$.
6.2. Online Adaptive Stochastic Dual Coordinate Ascent

See Algorithm 10 for details.

**Algorithm 10: OASDCA (Online Adaptive Stochastic Dual Coordinate Ascent)**

**Input:** \( \lambda > 0 \)

**Data:** \( \{(x_i, y_i)\}_{i=1}^{n} \)

**Initialize:** \( a^{1} = 0, w^{1} = 0, p_{i} = \frac{1 + \frac{1}{n^{2}}}{n + \sum_{j=1}^{n} \frac{1}{|y_j|}} \) or \( p_{i} = \frac{|x_{i}|}{\sum_{j=1}^{n} |y_j|}, c_i = 0, \) \( \forall i \in \{1, \ldots, n\}. \)

**for** \( t = 1, 2, \ldots, T \)

Sample \( i_{t} \) from \( \{1, \ldots, n\} \) based on \( p; \)

Calculate \( \Delta a_{i_{t}}^{t} \) using following formulas:

\[
\Delta a_{i_{t}}^{t} = \arg \max_{\Delta a_{i_{t}}} \left[ \frac{-\Delta a_{i_{t}}}{2} \|w^{t} + \frac{1}{n} \Delta a_{i_{t}} x_{i_{t}}\|^2 - \ell_{i_{t}}^{*} (- (a_{i_{t}}^{t} + \Delta a_{i_{t}})) \right];
\]

Set \( a_{i_{t}}^{t+1} \leftarrow a_{i_{t}}^{t} + \Delta a_{i_{t}}^{t} \);

Set \( w^{t+1} \leftarrow w^{t} + \frac{1}{n} \Delta a_{i_{t}}^{t} x_{i_{t}} \);

Calculate \( \sigma_{i_{t}} \leftarrow \ell(x_{i_{t}}^{T} w^{t}) + \ell^{*} (-a_{i_{t}}^{t}) + a_{i_{t}}^{t} x_{i_{t}}^{T} w^{t} \};

Set \( p_{i_{t}} \leftarrow \max \{\epsilon, \sigma_{i_{t}}\}; \)

**end**

**Output:** \( w^{T+1} \)
In this chapter, we discuss several issues regarding probabilities, ideas on sampling methods and the complexity of the algorithms.

## 7.1 Setting Probability

### 7.1.1 Minimum Probability

From our description of the adaptive sampling it is clear that for offline adaptive sampling (such as AdaSGD, AdaSDCA), assigning a minimum probability for each point is not necessary. Because even if a point was not sampled in an epoch, we still calculate its subgradient or gap and update the probability according to them. However, as wrote in Subsection 5.1.2, small probability has a potential of a bad behavior. Therefore, we choose Conservative Update for Adaptive SGDs. But for the online algorithm, it is essential to carefully set a small value as probability. We have a small, non-zero initial probability, which we shall denote by $\epsilon$ as shown in Definition 6.1.

### 7.1.2 Initial Probability

For offline adaptive sampling methods, the initial probability is far less important than the sampling strategies, because the probability distribution is updated at the end of each epoch. For non-adaptive sampling and online adaptive sampling methods, we need to choose the initial probability distribution, e.g. uniform distribution, or proportional to the norm of each point.

## 7.2 Sampling Methods

In this section, we offer three approaches to do the sampling in the algorithms.
7. Discussions

7.2.1 Naïve Approach

Given the probabilities, an efficient sampling method is to take the probabilities as the length of interval. For a probability distribution, the total length of the intervals is 1. We sample a number from $[0, 1]$, then find the corresponding index of the interval by sequentially deducting the length of interval. However, the average cost of this method is $O(n)$.

7.2.2 Binary Search

To improve on this, we precompute $s_i = \sum_{j=1}^{i} p_i$, with $s_0 = 0$ and $s_n = 1$ in $O(n)$ time. $s$ is an monotonic ascending array. We can find the corresponding index by performing a binary search on it, where each search can be done in $O(\log n)$ time.

However, this still requires a $O(n)$ update every time the probability distribution changes, which is still unacceptably slow. Therefore we introduce a new structure, the Segment Tree, to speed up the sampling procedure.

7.2.3 Segment Tree

As we can see from the above subsection, each update step for the probabilities costs $\Theta(n)$. For online algorithm, each iteration needs an operation of updating which is impractical. In this subsection, we introduce a new data structure for online adaptive algorithms.

The whole sampling procedure consists of three parts: precomputing, sampling and updating. Details are given as below.

- Precomputing the probabilities according to the initial probabilities
- Sampling a float number, and get the index of intervals that it falls in
- Updating the probabilities on-the-fly

The detailed explanation and pseudo code of implementation are given in the Appendix A.

For online adaptive algorithms, precomputing the ST before the algorithm starts costs $O(n \log n)$, the costs of sampling and updating per iteration are the same, namely $O(\log n)$. Segment Tree can also be applied to offline adaptive algorithms, where precomputing costs $O(n \log n)$ and sampling per iteration costs $O(\log n)$. In contrast, precomputing in binary search costs only $O(n)$. Therefore, we suggest using Segment Tree for online adaptive algorithms and Csiba-AdaSDCA+ while Binary Search is more suitable for offline adaptive ones.
7.3 Comparison of Complexity

Remark 7.1 \textit{nnz}: is the number of nonzero elements of the matrix consisting of all the samples in the dataset.

In the Table 7.1, the computational cost of an epoch consists only of training time, excluding the time taken to take samples and update the probability distribution.

AdaSGD and AdaSDCA take additional $k$ passes in each epoch. As we will mention in Chapter 8, if we choose $k = 1$ in our experiment, the cost of AdaSDCA is the same as AdaSDCA+.

AdaGrad needs to compute every entries of the feature in each iteration, so the cost per epoch is $nd$ where $d$ is the number of features. For sparse dataset, $nd \gg \text{nnz}$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>cost of an epoch</th>
<th>non-uniform</th>
<th>adaptive</th>
<th>online</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonUnifSGD</td>
<td>nnz</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NonUnifSDCA</td>
<td>nnz</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AdaSGD</td>
<td>$(k + 1)$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>AdaSVRG</td>
<td>$nd + k$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>$(k + 1)$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AdaSDCAS</td>
<td>$(k + 1)$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OASGD</td>
<td>nnz</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OASDCA</td>
<td>nnz</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AdaGrad</td>
<td>$2nd$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Csiba-AdaSDCA</td>
<td>$n$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Csiba-AdaSDCA+</td>
<td>$2$ nnz</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

7.3.1 Sampling and Probability Update

As shown in Section 7.2, the sampling cost is $O(\log n)$ per iteration. After each epoch, probability update takes at most $O(n \log n)$.

7.3.2 Total Computational Cost

The total computational cost includes training time, sampling time and additional precomputing or updating time. For example, suppose the size of dataset is $n$ and $k = 1$, the total cost per epoch is $2\text{nnz} + O(n \log n)$ for AdaSGD and AdaSDCA.
In this chapter, we evaluate the performance of all the proposed algorithms. First we will run some sanity-check experiments by finding the value of regularization parameter with minimum test error and verifying that the duality gap converges to 0. Then we will compare non-uniform algorithms with uniform algorithms. For the adaptive algorithms, we will first verify that SGD and SDCA converge to the same solution. We will then show how probabilities change in selected epochs, compare the performance of different values of k, study the Aggressive and Conservative strategies to update the probabilities and compare the two adaptive strategies for Adaptive SDCA. We then compare the performance of our algorithms with algorithms by other authors. Finally we will compare the online adaptive algorithms with non-adaptive algorithms.

8.1 Setup

8.1.1 Training

We train an SVM on the training dataset using each of the above algorithms, then evaluate the solution by applying on the test dataset. We use the two datasets as shown in Table 8.1. rcv1 is a corpus of Reuters news stories and astro-ph is astronomy data.

Epoch

By epoch, we mean $n$ consecutive iterations, where $n$ is the number of samples.

Parameters

the parameters used for training are as follows:
8. Experimental Results

1. $\lambda$: the regularization parameter, varying from $10^{-10}$ to 1 and differing by a factor of 10. We fix $\lambda$ at the value which minimizes training error (see Subsection 8.1.2)

2. $k$: the parameter in AdaSGD and AdaSDCA, which indicates how many iterations per epoch we use to evaluate performance of recent $w$ (see results in Subsection 8.3.3)

3. #epoch: the number of epochs

4. #rounds: the number of times we run an algorithm over one dataset with the same number of epochs such that we can average out the running results to reduce the variance in performance. Generally, people pick from 5 to 20. Accordingly, we fix it to 10 throughout this chapter

Performance Metric

1. Total running time: total running time including training time, sampling time etc..
2. Training time: training time without sampling time
3. Objective calculation time: time for calculating objective
4. Norm value: L2-norm of $w^T$
5. Loss value: value of $\ell(w^T)$
6. Zero-one error: zero-one error on the test dataset
7. Objective value: value of $f(w^T)$
8. Test loss: sum of loss on test dataset
9. Test error: error on test dataset

Definition 8.1 The primal sub-optimality of algorithm is defined as $P(w(\alpha)) - P(w^*)$. The dual sub-optimality of SDCA algorithm is defined as: $D(\alpha^*) - D(\alpha)$.

In this thesis, we use primal suboptimality and test error to evaluate the performance of each algorithm. All the figures about primal suboptimality are plotted using the log scale.

The datasets we used are listed in Table 8.1, including the size of training set, the size of test set, the number of features, sparsity and the norm variance of samples. $\lambda$ is chosen according to the rule that it should be close to $1/n$ [15]. In Subsection 8.1.2, we perform experiments to find the best $\lambda$ with which the test error yielded is the smallest.
8.2 Non-Uniform Probability

8.2.1 Initial Probabilities for Non-Uniform Sampling

We tested the following methods to initialize our probability distribution:

8.1.2 Finding the Regularization Parameter $\lambda$ which Minimizes Test Error

In this subsection, we do grid search to find the best $\lambda$ which yields the smallest test error in order to avoid over-fitting. The value of optimal $\lambda$ will be used in all the experiments in the following sections.

Here we run 200 epochs and average out the results of 10 rounds for uniform SDCA. In Table 8.2 and 8.3, we can see that the optimal $\lambda$ is 0.001 for rcv1 and astro-ph.

After obtaining the values of optimal $\lambda$ for each dataset, we only use them in the subsequent experiments. The optimal $\lambda$ is consistent with that obtained by other algorithms, because all the algorithms will converge to the same value of test error (see Subsection 8.3.1 for alignment checking). We take the optimal primal objective value from this experiment, which is 0.0686384 on rcv1 and 0.0628627 on astro-ph. By Definition 8.1, when we plot the primal suboptimality in subsequent figures, we use a value slightly smaller than the above values for $P(w^*)$ to avoid log(0).

Trivially, we can see that $\lambda$ value with the smallest test error does not correspond to the $\lambda$ value with minimum primal objective value, because a small $\lambda$ leads to a small regularizer.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training(n)</th>
<th>Test</th>
<th>Features (d)</th>
<th>Sparsity($\frac{nnz}{nd}$)</th>
<th>Norm Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcv1</td>
<td>20,242</td>
<td>677,399</td>
<td>47,236</td>
<td>0.16%</td>
<td>8.57034</td>
</tr>
<tr>
<td>astro-ph</td>
<td>29,882</td>
<td>32,487</td>
<td>99,757</td>
<td>0.08%</td>
<td>3.30349</td>
</tr>
</tbody>
</table>

| Table 8.2: Test error with different values of $\lambda$ on dataset rcv1 |
|----------------|---------------------|----------------|-----------------|-----------------|----------------|
| $\lambda$      | 1e-1                | 1e-2            | 5e-3            | 1e-3            | 5e-4           | 1e-4          | 1e-5          |
| Test Error      | 0.07005             | 0.05160         | 0.04833         | **0.04713**     | 0.04913        | 0.05693       | 0.06011       |

| Table 8.3: Test error with different values of $\lambda$ on dataset astro-ph |
|----------------|---------------------|----------------|-----------------|-----------------|----------------|
| $\lambda$      | 1e-1                | 1e-2            | 5e-3            | 1e-3            | 5e-4           | 1e-4          | 1e-5          |
| Test Error      | 0.07335             | 0.04103         | 0.03715         | **0.03441**     | 0.03586        | 0.04371       | 0.04786       |

8.2 Non-Uniform Probability
8. Experimental Results

1. \[ \frac{\|x_i\| + \sqrt{\lambda}}{\sum_{j=1}^{n|\|x_j\|}} \]
2. \[ \frac{\|x_i\|}{\sum_{j=1}^{n|\|x_j\|}} \]
3. \[ \frac{1/\|x_i\|}{\sum_{j=1}^{n|\|x_j\|}} \]

As we can see from (4.17) and (5.1), the probabilities are related to the norm because of the solution for the optimization problem.

We use the \( \lambda \) value from Subsection 8.1.2. We do not put a figure here because all the experiments show that there is no significant difference between the first two methods, while the third method behaves badly. In the following experiments, we always use method 1 for initialization.

8.2.2 Verifying the Convergence of Duality Gap

We conduct experiments to validate the convergence of duality gap for both datasets. From the Tables 8.4 and 8.5, we can see that the duality gap converges to 0 in later epochs. It converges faster in astro-ph than in rcv1.

<table>
<thead>
<tr>
<th>#epoch</th>
<th>average duality gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0863765</td>
</tr>
<tr>
<td>3</td>
<td>0.0105347</td>
</tr>
<tr>
<td>10</td>
<td>1.7485e-04</td>
</tr>
<tr>
<td>20</td>
<td>2.21547e-05</td>
</tr>
<tr>
<td>50</td>
<td>3.12797e-06</td>
</tr>
<tr>
<td>100</td>
<td>5.47897e-07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#epoch</th>
<th>average duality gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0883917</td>
</tr>
<tr>
<td>3</td>
<td>6.13163e-03</td>
</tr>
<tr>
<td>10</td>
<td>3.93673e-05</td>
</tr>
<tr>
<td>20</td>
<td>6.24779e-07</td>
</tr>
<tr>
<td>50</td>
<td>6.7474e-10</td>
</tr>
<tr>
<td>100</td>
<td>1.43083e-12</td>
</tr>
</tbody>
</table>

8.2.3 Comparison of Uniform Sampling with Non-Uniform Sampling

In this subsection, we compare the performance of uniform sampling with non-uniform sampling. Figure 8.1 shows that there is no big difference be-
8.3 Adaptive Probabilities

8.3.1 Verifying the Alignment of SGDs and SDCAs

Definition 8.2 The alignment of two solutions $w$ and $w'$ is given by $\frac{\langle w, w' \rangle}{\|w\| \|w'\|}$.

We run the experiment with 200 epochs for SDCAs and 2000 epochs for SGDs. We collect the final value of $w$, calculate the alignment and show the results in Table 8.6. All the values of alignment are close to 1, so we can conclude that all the algorithms converges to a same value of $w$.

8.3.2 Histogram of Probability Changes

We analyze the sampling probabilities of AdaSGD and AdaSDCA for each epoch and plot the probability distribution for the selected epochs.

On Figure 8.2 and 8.4, we can see that by running AdaSGD, the probabilities mostly fall into one bin of the histogram after the first epoch, which means that most of the points are correctly classified and they are allocated with the same probability by Algorithm 4.

In Figure 8.3 and 8.5, we observe a similar pattern for AdaSDCA.

In AdaSDCA (Figure 8.5), compared to AdaSGD (Figure 8.4):

1. Most points have probability equal to zero
2. There are more points in the higher-probability bins because all the probabilities sums up to 1
8. Experimental Results

Figure 8.2: Histogram of sampling probabilities changed by AdaSGD in selected epochs on rcv1
Figure 8.3: Histogram of sampling probabilities changed by AdaSDCA in selected epochs on rcv1
Figure 8.4: Histogram of sampling probabilities changed by AdaSGD in selected epochs on astro-ph
Figure 8.5: Histogram of sampling probabilities changed by AdaSDCA in selected epochs on astro-ph
8. Experimental Results

Table 8.6: Pairwise alignment of SGDs and SDCAs on rcv1

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Algorithm Name</th>
<th>Value of alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaSGD</td>
<td>AdaSDCA</td>
<td>0.999933</td>
</tr>
<tr>
<td>AdaSGD</td>
<td>NonUnifSGD</td>
<td>0.999834</td>
</tr>
<tr>
<td>AdaSGD</td>
<td>NonUnifSDCA</td>
<td>0.999927</td>
</tr>
<tr>
<td>AdaSGD</td>
<td>Uniform SGD</td>
<td>0.999919</td>
</tr>
<tr>
<td>AdaSGD</td>
<td>Uniform SDCA</td>
<td>0.999857</td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>NonUnifSGD</td>
<td>0.999766</td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>NonUnifSDCA</td>
<td>0.999883</td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>Uniform SGD</td>
<td>0.999895</td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>Uniform SDCA</td>
<td>0.999918</td>
</tr>
<tr>
<td>NonUnifSGD</td>
<td>NonUnifSDCA</td>
<td>0.999770</td>
</tr>
<tr>
<td>NonUnifSGD</td>
<td>Uniform SGD</td>
<td>0.999865</td>
</tr>
<tr>
<td>NonUnifSGD</td>
<td>Uniform SDCA</td>
<td>0.999875</td>
</tr>
<tr>
<td>NonUnifSDCA</td>
<td>Uniform SGD</td>
<td>0.999894</td>
</tr>
<tr>
<td>NonUnifSDCA</td>
<td>Uniform SDCA</td>
<td>0.999912</td>
</tr>
<tr>
<td>Uniform SGD</td>
<td>Uniform SDCA</td>
<td>0.999878</td>
</tr>
</tbody>
</table>

According to the figures, AdaSGD has less variance in probability distribution and hence it is less biased. This confirms the characteristic of Conservative Update.

8.3.3 Performance of Different Values of $k$

In this subsection, we run AdaSGD and AdaSDCA with different values of parameter $k$ from 1 to 7. In Figure 8.6, we can see that different values of $k$ do not make any significant difference to the performance of AdaSGD, while larger values of $k$ improve the performance for AdaSDCA slightly. However, $k$ additional passes per epoch cost much more in adaptive algorithms than in non-adaptive algorithms. Because of the tradeoff between the performance of running time, we use $k = 1$ for later experiments.

8.3.4 Performance of Two Updating Algorithms

We proposed Aggressive Update (Algorithm 3) and Conservative Update (Algorithm 4) for changing probabilities for adaptive algorithms in Subsection 5.1.2. Here we compare the performance of the two updating algorithms in Figure 8.7. Conservative Update works much better than Aggressive Update on AdaSGD, which confirms our observation in (5.2). On the contrary, Aggressive Update performs better than Conservative Update on AdaSDCA. Therefore, we use Conservative Update for AdaSGD and Aggressive Update for AdaSDCA in subsequent experiments.
8.3. Adaptive Probabilities

8.3.5 Different Adaptive Strategies for AdaSDCA

We propose two adaptive strategies for SDCA as described in Subsection 5.3.1. One is related to Duality Gap and the other is Subgradient with Conservative update. We compare their performance in Figure 8.8. As discussed in Subsection 5.3.2, duality gap is a more natural way to evaluate the performance of $w$. Therefore it performs better than subgradient as expected.

8.3.6 Comparison of Adaptive Algorithms

We choose $m \in \{2, 10, 50\}$ as recommended in [3] for AdaSDCA+. The performance is exactly the same when we use small $m$, while it is relatively worse when we use larger $m$ such as $m = 50$.

We run five adaptive algorithms: AdaSGD, AdaGrad, AdaSDCA, AdaSVRG and AdaSDCA+. The results can be seen in Figure 8.9. In terms of primal suboptimality, AdaSDCA has the best performance among the five algorithms. It is followed by AdaSVRG and AdaSGD. AdaSVRG has a slightly better performance than AdaSGD. AdaSDCA+ has a same performance as
8. Experimental Results

AdaSDCA in the very early epochs, but the objective value does not decrease anymore after that. AdaGrad performs the worst among the all. In terms of test error, the worst is AdaGrad while the best is AdaSVRG. AdaSGD is close to AdaSVRG. They are followed by AdaSDCA and AdaSDCA+. The test error of these four algorithms are close. In this case, in order to illustrate the difference better, we plot in Figure 8.10 the performance of AdaSDCA+, AdaSDCA and AdaSGD.

In Figure 8.10, we plot the performance of three algorithms on astro-ph. AdaSDCA has the best performance both on primal suboptimality and test error while AdaSDCA+ has the worst performance.

8.3.7 Comparison of Adaptive Algorithms with Non-Adaptive Algorithms

In this subsection, we compare the detailed training time and total running time in Table 8.7 and 8.8. We average out the results of 500 epochs on SGD algorithms and 50 epochs on SDCA algorithms. We do not put UnifSGD
8.3. Adaptive Probabilities

Figure 8.10: Comparison of three adaptive algorithms on astro-ph

and UnifSDCA to compare with other algorithms in the tables and figures because the performance is approximately the same as their non-uniform counterparts.

Table 8.7: Detailed training time and total running time per epoch on rcv1

<table>
<thead>
<tr>
<th></th>
<th>rcv1</th>
<th>Average training time(s)</th>
<th>Average total running time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaSGD</td>
<td>0.04765</td>
<td>0.2059</td>
<td></td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>0.05042</td>
<td>0.2064</td>
<td></td>
</tr>
<tr>
<td>NonUnifSGD</td>
<td>0.04244</td>
<td>0.1988</td>
<td></td>
</tr>
<tr>
<td>NonUnifSDCA</td>
<td>0.04716</td>
<td>0.2037</td>
<td></td>
</tr>
</tbody>
</table>

Notice that from Table 8.7 and 8.8, adaptive algorithms cost slightly more than non-adaptive algorithms in terms of both training time and total running time (which includes sampling time) when $k = 1$. Theoretically, adaptive algorithms take one more pass than non-adaptive algorithms according to Section 7.3.

Table 8.8: Detailed training time and total running time per epoch on astro-ph

<table>
<thead>
<tr>
<th></th>
<th>astro-ph</th>
<th>Average training time(s)</th>
<th>Average total running time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaSGD</td>
<td>0.07236</td>
<td>0.1363</td>
<td></td>
</tr>
<tr>
<td>AdaSDCA</td>
<td>0.07050</td>
<td>0.1343</td>
<td></td>
</tr>
<tr>
<td>NonUnifSGD</td>
<td>0.06284</td>
<td>0.1259</td>
<td></td>
</tr>
<tr>
<td>NonUnifSDCA</td>
<td>0.07054</td>
<td>0.1339</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.11 illustrates the change of primal suboptimality with respect to the total running time. Our objective value here is chosen by the optimality reached by NonUnifSGD after 500 epochs. AdaSDCA and NonUnifSDCA quickly reach the objective value and AdaSGD takes only half of the time NonUnifSGD takes. We list the corresponding number of epochs in Table 65.
8. Experimental Results

Table 8.9: The number of epochs taken to reach the same level of optimality

<table>
<thead>
<tr>
<th></th>
<th>rcv1</th>
<th>AdaSDCA</th>
<th>NonUnifSDCA</th>
<th>AdaSGD</th>
<th>NonUnifSGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>#epochs</td>
<td>9</td>
<td>35</td>
<td>210</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>astro-ph</th>
<th>AdaSDCA</th>
<th>NonUnifSDCA</th>
<th>AdaSGD</th>
<th>NonUnifSGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>#epochs</td>
<td>8</td>
<td>28</td>
<td>195</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

8.9.

![Graph](image1)

Figure 8.11: Comparison of the total running time to reach the same optimality

In Figure 8.12, we compare the number of vector operations needed (without sampling) to reach the same primal suboptimality for the four algorithms.

![Graph](image2)

Figure 8.12: Comparison of the vector operations taken to reach the same optimality

Figure 8.13 and 8.14 demonstrates primal suboptimality and test error rate in different epochs. On rcv1, AdaSGD and NonUnifSGD have lower test error than AdaSDCA and NonUnifSDCA while on astro-ph, AdaSDCA and NonUnifSDCA have lower test error. In terms of primal suboptimality, we observe similar trends in both figures. In either SDCA or SGD, adaptive algorithm is always better than non-uniform algorithm.
8.4 Online Algorithms

8.4.1 Online Adaptive Algorithms vs. Non-Adaptive Algorithms

In this subsection, we run OASGD and OASDCA and compare their performance to NonUnifSGD and NonUnifSDCA.

From Figure 8.15 and 8.16, OASDCA’s performance is almost the same as NonUnifSDCA. However, OASGD’s performance is worse than that of NonUnifSGD. It is hard to design online adaptive algorithms that outperforms non-adaptive algorithms, so we leave it to future work.

8.5 Summary

In this section, we summarize the experimental results.

- We chose $\lambda = 0.001$ for both rcv1 and astro-ph.
- The performance is almost the same for uniform and non-uniform sampling on both datasets.
8. Experimental Results

We compare the performance of Conservative Update and Aggressive Update on AdaSGD and AdaSDCA. Conservative Update works better on AdaSGD while Aggressive Update works better on AdaSDCA.

AdaSDCA (adaptive algorithm with duality gap) performs better than AdaSDCAS (adaptive algorithm with subgradients).

AdaSDCA has the best performance among all the adaptive algorithms (AdaSDCA, AdaSGD, AdaSVRG, AdaGrad and AdaSDCA+) and AdaSGD is the second best.

AdaSVRG achieves a slightly better performance per epoch than AdaSGD but sacrifices the running time on sparse datasets.

To reach the same optimality given by 500 epochs run on NonUnifSGD, AdaSGD takes only around 200 epochs, whereas NonUnifSDCA takes around 30 which is three times more than AdaSDCA does.

OASDCA performs almost the same as NonUnifSDCA while OASGD performs much worse than NonUnifSGD.
Chapter 9

Conclusion

In this thesis, we discussed the non-uniform and adaptive sampling strategies for the stochastic optimization problems, using SGD and SDCA as examples. We gave a novel analysis of the convergence rate for SGD by adopting a new weighting technique. We studied adaptive sampling and analyzed convergence rate on both SGD and SDCA for the adaptive variants. Our analysis showed that adaptive sampling reduces the number of iterations taken to reach same optimality compared to non-uniform sampling. We also modified the offline adaptive algorithms to provide the online versions.

We offered the pseudo code for all the algorithms as well as the data structure for sampling and conducted experiments to validate our theories. The results showed that both AdaSGD and AdaSDCA reach the same level of optimality in fewer epochs than their uniform and non-uniform counterparts without sacrificing the running time. In comparison to other existing algorithms, AdaSGD outperforms AdaGrad and AdaSDCA outperforms Csiaba-AdaSDCA+.
Appendix A

Segment Tree

The Segment Tree (ST) is a tree data structure that stores information of intervals for querying. Each tree node with id $\nu$ is an interval denoted as $[a, b]$, here $a$ is the left-end of the interval and $b$ is the right-end of the interval. In our case, it stores the summation of probabilities from $p_a$ to $p_b$.

The non-leaf node $[a, b]$ has its left child node $[a, c]$ and right child node $[c + 1, b]$ where $c = \lfloor (a + b) / 2 \rfloor$. For each leaf node, $a = b$. For tree node with id $\nu$, we denote the id of its left child node as $2\nu + 1$ and the id of its right child node as $2\nu + 2$.

For the index $i$, the update operation is done by traversing along the path from the leaf node $[i, i]$ to the root node $[1, n]$. The height of the segment tree is $\log n$, so the time complexity of update is $O(\log n)$.

Each tree node is represented by a structure of (sum, left, right), which is given by:

- **sum**: summation of probability in current interval
- **left**: left-end of the interval
- **right**: right-end of the interval

**Construction** Given initial probabilities $p$, the segment tree can be constructed in a recursive way as shown in Algorithm 11. If a node is a leaf node, we assign the corresponding probability, otherwise its value can be
computed by adding \( \text{sum} \) in left child node and \( \text{sum} \) in right child node.

**Algorithm 11: Segment Tree Construction**

```plaintext
function \textsc{Construct}(\text{TreeNum } \nu, \text{leftEnd } le, \text{rightEnd } re)
    Set \( m \leftarrow \lfloor \text{le + re} \rfloor / 2; \)
    if \( m < re \) then
        \textsc{Construct}(2 * \nu + 1, \text{le}, \text{m});
        \textsc{Construct}(2 * \nu + 2, \text{m + 1}, \text{re});
        \text{Set Tree[\nu].sum} \leftarrow \text{Tree[2 * \nu + 1].sum} + \text{Tree[2 * \nu + 2].sum};
    end
    else
        \text{Set Tree[Num].sum} \leftarrow p_m;
    end
end function
```

**Query** Similarly as before, we conduct search recursively on the tree for retrieving the summation probability from \( p_1 \) to \( p_i \). In order to get the value, we need to call \textsc{Query}(0, 1, i). For a query interval \([a, b]\) of a tree, there are three cases. The interval will lie entirely within the range of left child node, or entirely on the range of right child node, or in both (ranges). The total complexity of \textsc{Query} is \( O(\log n) \).

**Algorithm 12: Segment Tree Query**

```plaintext
function \textsc{Query}(\text{TreeNum } \nu, \text{leftEnd } le, \text{rightEnd } re)
    Set \( m \leftarrow \lfloor \text{Tree[\nu].left + Tree[\nu].right} \rfloor / 2; \)
    if \( \text{Tree[\nu].left} > m \) then
        return \textsc{Query}(2 * \nu + 2, \text{le}, \text{re});
    end
    else if \( \text{Tree[\nu].right} < m \) then
        return \textsc{Query}(2 * \nu + 1, \text{le}, \text{re});
    end
    else
        return \textsc{Query}(2 * \nu + 1, \text{le}, \text{m}) + \textsc{Query}(2 * \nu + 2, \text{m + 1}, \text{re});
    end
end function
```

**Update** Update can be performed by \textsc{Update}(0, \( i \), \( \delta \)) where \( i \) is the index of current sampled point and \( \delta \) is the value of change in probability. From Algorithm 12 we can see that it is also a recursive function. The maximum number of tree nodes to be updated is \( 2 \log n \), so the time complexity for
Query is $O(\log n)$.

Algorithm 13: Segment Tree Update

```plaintext
function Update(TreeNum ν, Index i, Change δ)
    if Tree[ν].left = Tree[ν].right and Tree[ν].left = i then
        Set Tree[ν].sum += δ;
    end
    else
        Set m to (Tree[ν].left + Tree[ν].right) / 2;
        if i > m then
            return Update(2 * ν + 2, i, δ);
        end
        else
            return Update(2 * ν + 1, i, δ);
        end
        Set Tree[ν].sum ← Tree[2 * ν + 1].sum + Tree[2 * ν + 2].sum;
    end
end function
```
Appendix B

Implementation Details

We implement all the algorithms in C++11, based on Shai Shalev-Shwartz’s Pegasos code, which can be download here\(^1\).

My implementation can be downloaded at this link\(^2\). It runs on the machine with 16GB memory and Intel Core i5-4570 CPU. All the datasets can be easily found by Google or from this site\(^3\).

In this thesis, all the figures are plotted by matplotlib\(^4\) in Python.

---

\(^1\)http://www.cs.hui.ac.il/\texttt{~}shais/code/pegasos.tgz
\(^2\)https://github.com/raynald/master-thesis-imple
\(^3\)http://www.csie.ntu.edu.tw/\texttt{~}cjlin/libsvmtools/datasets/binary.html
\(^4\)http://matplotlib.org
Appendix C

Glossary

AE  Adaptive Encode
ERM  Empirical Risk Minimization
LASSO  Least Absolute Shrinkage and Selection Operator
LR  Logistic Regression
RR  Ridge Regression
SDCA  Stochastic Dual Coordinate Ascent
SGD  Stochastic Gradient Descent
ST  Segment Tree
SVM  Support Vector Machine
SVRG  Stochastic Variance Reduced Gradient
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

ADAPTIVE PROBABILITIES IN STOCHASTIC OPTIMIZATION ALGORITHMS

Author(s) (in block letters):

ZHONG

First name(s):

LEI

Name(s):

ZHONG

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

ZÜRICH, 10-04-2015

Signature(s)