# Gradient Descent for Optimization Problems With Sparse Solutions

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th>Citation</th>
<th>Chen, Hsieh-Chung. 2016. Gradient Descent for Optimization Problems With Sparse Solutions. Doctoral dissertation, Harvard University, Graduate School of Arts &amp; Sciences.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:33493549">http://nrs.harvard.edu/urn-3:HUL.InstRepos:33493549</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Gradient Descent for Optimization Problems with Sparse Solutions

A DISSERTATION PRESENTED
BY
HSIEH-CHUNG CHEN
TO
SCHOOL OF ENGINEERING AND APPLIED SCIENCES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
COMPUTER SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
MAY 2016
Gradient Descent for Optimization Problems with Sparse Solutions

Abstract

Sparse modeling is central to many machine learning and signal processing algorithms, because finding a parsimonious model often implicitly removes noise and reveals structure in data. They appear in applications such as feature selection, feature extraction, sparse support vector machines, sparse logistic regression, denoising, and compressive sensing. This raises a great interest in solving optimization problems with sparse solutions.

There has been substantial interest in sparse optimization in the last two decades. Out of the various approaches, the gradient descent methods and the path following methods have been most successful. Existing path following methods are mostly designed for specific problems. Gradient descent methods are more general, but they do not explicitly leverage the fact that the solution is sparse.

This thesis develops the auxiliary sparse homotopy (ASH) method for gradient descent, which is designed to converge quickly to answers with few non-zero components by maintaining sparse interim state while making sufficient descent. ASH modifies gradient methods by applying an auxiliary sparsity constraint that relaxes dynamically overtime. This principle is applicable to general gradient descent methods, in-
including accelerated proximal gradient descent, coordinate descent, and stochastic gradient descent.

For sparse optimization problems, ASH modified algorithms converge faster than the unmodified counterparts, while inheriting their convergence guarantees and flexibility in handling various regularization functions. We demonstrate the advantages of ASH in several applications. Even though some of these problems (notably LASSO) have attracted many dedicated solvers over the years, we find that ASH is very competitive against the state-of-the-art for all these applications in terms of convergence speed and cost per-iteration.
# Contents

0 Introduction ........................................ 1
   0.1 Road map ......................................... 5

1 Sparse Optimization Problems ................... 6
   1.1 Fitness Function .................................... 7
   1.2 Sparse Regularization .............................. 8
   1.3 Smoothness ......................................... 12
   1.4 Convexity ........................................... 13
   1.5 Examples of Cost Functions for Different Applications .............. 13
      1.5.1 Feature Selection with LASSO ....................... 13
      1.5.2 Sparse Coding and Feature Extraction ............................. 14
      1.5.3 Compressive Sensing .................................. 14
      1.5.4 Classifier ......................................... 15
   1.6 Summary and Remarks .............................. 15

2 Gradient Methods ...................................... 17
   2.1 Gradient Descent .................................. 18
   2.2 Proximal Gradient Descent (PG) ...................... 19
      2.2.1 Evaluating the Proximal Operator .................... 21
      2.2.2 Selecting Step Size .................................. 22
2.2.3 Accelerated Proximal Gradient Descent (APG) ............... 22
2.2.4 Accelerated Proximal Gradient for Non-convex Optimization ... 24
2.2.5 Generalization to Second Order Methods ..................... 24
2.2.6 Generalization to Mirror Descent ............................. 26
2.3 Coordinate Descent (CD) ........................................ 26
2.4 Stochastic Gradient Descent (SGD) .............................. 28
2.5 Summary and Remarks ............................................ 29

3 HOMOTOPY AND WORKING SET METHODS .......................... 31
  3.1 Least Angle Regression (LARS) ................................. 32
  3.2 Feature-Sign Search (FSS) ...................................... 33
  3.3 Proximal Gradient Homotopy (PGH) ............................ 34
  3.4 Matching Pursuit Lasso (MPL) .................................. 35
  3.5 Orthogonal Matching Pursuit (OMP) ............................ 36
  3.6 Summary and Remarks ............................................ 37

4 AUXILIARY SPARSE HOMOTOPY (ASH) ................................. 38
  4.1 Intuition ......................................................... 38
  4.2 A Working Set Approach ....................................... 39
  4.3 A Homotopy Approach .......................................... 40
  4.4 The Auxiliary Sparse Homotopy Method (ASH) ................. 42
    4.4.1 ASH for Separable Regularization .......................... 43
    4.4.2 ASH for Coordinate Descent ............................... 46
  4.5 Characteristics of ASH .......................................... 47
  4.6 Summary and Remarks ............................................ 48
5 Applications and Evaluations 50

5.1 The LASSO Problem ................................................. 51
5.2 Sparse Coding in Hierarchical Models ....................... 54
  5.2.1 Example: Emotion Classification ......................... 58
5.3 Compressive Sensing ............................................. 60
  5.3.1 Solving the Compressive Sensing Problem ................. 61
  5.3.2 Nonlinear compressive sensing ............................. 62
5.4 Sparse Logistic Regression ....................................... 65
5.5 Sparse Neural Nets ............................................... 67
5.6 Summary and Remarks ............................................ 69

6 Discussion 72

6.1 Brief Summary of Different Methods .......................... 72
6.2 Computational Gains from Sparsity ............................ 74
6.3 Parallelizability ................................................... 75
6.4 Benefit of \ell_0-Sparse Solutions in Non-convex Problems .... 76
6.5 Summary and Remarks ............................................ 76

7 Conclusion 78

References 87
Dedicated to my brother.
Acknowledgments

First, I want to thank Doug for introducing me to HT. HT is an incredible advisor that leads by example. He is so enthusiastic about research, and fearless in attacking all kinds of problems. Most importantly, he is excited about new ideas all the time, which made it very fun to work with him.

I would like to thank all members of Kung lab, especially Dario and CK who guided me in my first years when I had zero clue about wireless communication; Tsung-Han and Steve whom I rely on whenever I have some half-baked ideas; Michael and Surat for being real hackers that made our projects work; Brad, for making me not the only introvert. Many thanks to my committee Shlomo and Na Li for helping out with my thesis; and to Harvard in general for providing such a great place for a graduate student.

To my friends who helped me set up home in this foreign place: Amanda, Hai-Yin, chhsiao, Hsiao-Han, Hannah, and my dear roommates Ching-Fu and Meredith. I will always be filled with nostalgia whenever I look back to the time we spent together.

To Mom and Dad: Thank you for being my friends, consultants, Dota buddy and being so supportive in every stage of my life.

To my wife: thank you for being my wife.
Introduction

**Sparse Optimization**

Optimization is at the core of machine learning and signal processing — it is the tool that we use to implement new models for large scale problems. A model can be described by its parameters, and the parameters are normally obtained through solving optimization problems. There is a wide range of applications that rely heavily on efficient optimization tools. For example: feature selection [67], feature extraction [76], representation learning [9], denoising [26], in-painting [64], compressive sensing [16, 23], and neural networks [35].

Enforcing parameters to be sparse is a very effective way to obtain parsimonious
models or data representations\(^1\). Intuitively, sparsity constrains the solution space in a non-trivial way (unlike limiting the number of parameters), which gives us the freedom to use complex models with many of parameters without over-fitting to data.

Sparse modeling has proven to be useful for many of the applications listed above. In image classification, we can obtain high level representations of images by encoding them with a few prominent features [20]. In compressive sensing, we can restore signals from underdetermined systems by finding sparse solutions [23]. Sparse models are also used in the studies of genetic disorders, because only a few genes are expected to be related to a given disease [19]

STATE OF THE FIELD

Sparse optimization problems have attracted substantial attention over the last two decades, partly because sparse models are effective in many applications, and partly because the sparse structures in optimization problems are interesting in their own right. For example, the famous \textit{least squares with} \(\ell_1\) \textit{penalty} (LASSO) problem has attracted hundreds of dedicated solvers, together attacking the problem through many different angles. Despite the substantial amount of existing work, solving sparse optimization problems remains a very active field of research. Gradient descent and path-following methods are the most competitive among existing solvers, they will be the focus in the remainder of this thesis.

Gradient descent methods are reliable and general, in the sense that they always converge to a locally optimal solution under mild assumptions. Furthermore, they can exploit composite structures in optimization problems through the use of a proximal operator to speed-up convergence. Proximal gradient descent has been shown to be

\(^1\)We say that a set of parameters is sparse if only a few of them are nonzero.
competitive for solving optimization problems with sparse regularization. Another advantage of gradient descent is that many of its variations can be explained under a general framework.

Homotopy and working set methods are designed to leverage the assumption that the solutions are sparse. They both follow a sparse path toward the final solution in the sense that all interim solutions are sparse. Working set methods slowly expands the support set, and iteratively solves subproblems with few variables until the solution is found. Homotopy methods starts out with strong sparse regularization so that solutions can be found very quickly, and then gradually relax the regularization until the desired solution is reached. Most of these path-following methods are designed for specific problems.

**Contribution**

This thesis focus on the design of a path-following strategy for gradient descent methods. Typical gradient descent methods only rely on local approximation of the cost function, which does not take into account the fact that the solution is sparse. However, they can be made to explicitly search for sparse solutions, as shown in Figure 1.

The main challenge is to maintain the generality of gradient descent methods while taking advantage of the additional knowledge. Our solution is based on the following core ideas:

1. **Homotopy by introducing an auxiliary sparsity constraint.**

   Unlike typical working set method, we can handle the same set of cost functions that can be solved using a standard gradient descent method. The path-

---

2Support set is the set of coefficients that are nonzero.
following strategy only assumes that the solution is \( \ell_0 \) sparse, and does not depend on specific regularization functions.

2. **Support set expansion based on gradient descent progression.**

By leveraging information gathered in the gradient steps, we can have an adaptive support set expansion strategy that is parameter-free.

The resulting method is called the *auxiliary sparse homotopy* (ASH) method, which inherits the robustness of gradient methods and the computational advantages of typical path-following methods. We show that ASH can be applied to many gradient descend methods to improve both their convergence and per-iteration computation cost. Moreover, the improved methods consistently outperform state-of-the-art by a large margin.

![Graph](image)

**Figure 1:** For an \( \ell_1 \) box-constraint penalty, blue arrows illustrate the solution path produced by limiting support set expansion, contrasted with the less-direct red path of standard gradient methods. Intuitively, convergence to a sparse minimizer would be faster if extraneous directions are avoided from the beginning, rather than being refined away in later iterations.
0.1 Road map

Section 1 introduces the general sparse optimization problem, and highlights some special cases that arises from various applications. The next two sections covers the two types of approaches that are currently most competitive in solving sparse optimization problems. Section 2 provides a walk-through of gradient methods using a unified view. Section 3 describes working set and homotopy methods. We present the ASH method in Section 4, and provide two different interpretations of ASH. Section 5 presents several applications that involves sparse optimization, and demonstrates the performance of ASH on these applications. We discuss the strengths and weaknesses of different methods in Section 6, and make our conclusion in Section 7.
Sparse Optimization Problems

“Essentially, all models are wrong, but some are useful.”

— George E. P. Box

Occam’s razor is a useful principle in many areas of science: Among competing models that explains a given phenomenon, the simplest one should be preferred over the more complicated. Assuming that the complexity of a model is proportional to the number of parameters in it, one effective way to obtain parsimonious models is through sparse regularization of the parameters. Intuitively, having the sparsity regularization means that we are searching simple models within a large solution space.

Sparse modeling has been proven to be useful in many machine learning and signal
processing applications. In image classification, encoding image patches with a few prominent features yields high level representations that are easier to classify [20]. In compressive sensing, signals can be uniquely restored from underdetermined systems by restricting the solution to be sparse [23]. In the studies of genetic disorders, sparse models are used because only a few genes are expected to be related to a given disease [19].

All models are trained by fitting parameters to some given data, so it is important to have the right optimization tools. In this thesis, we are interested in designing efficient solvers for the following general optimization problem:

\[
\min_{z \in \mathbb{R}^n} \{Q(z) = f(z) + \lambda \cdot g(z)\}
\]  

(1.1)

where \( f : \mathbb{R}^n \to \mathbb{R} \) is a differentiable fitness function, and \( g : \mathbb{R}^n \to \mathbb{R} \) is a sparsity-promoting regularization function that is not necessarily smooth or differentiable. The parameter \( \lambda \geq 0 \) controls the strength of the regularization term. We assume that most of the coefficients in the solution are zeros.

1.1 Fitness Function

Function \( f \) in (1.1) is called the fitness function. In the context of supervised learning, we are given a set of \( N \) samples \( \{x_i, y_i\}^N \) as training data, and the goal is to use a certain model \( \mathcal{M} \) with parameters \( z \) to predict \( y \) when given a new \( x \). Learning the parameters \( z \) is an optimization problem where we try to minimize the loss between prediction \( \hat{y} = \mathcal{M}_z(x_i) \) and the ground truth \( y_i \) for the training set, which can be
expressed as:

$$
\min_z \left\{ f(z) \equiv \frac{1}{N} \sum_{i} \text{loss}(M_z(x_i), y_i) \right\}
$$

(1.2)

For example, the loss function can be squared distance $\text{loss}(\hat{y}, y) = (\hat{y} - y)^2$ for least squares regression, the logistic loss $\text{loss}(\hat{y}, y) = \log(1 + \exp(\hat{y}y))$ for logistic regression, or hinge loss $\text{loss}(\hat{y}, y) = (1 - \hat{y}y)_+$ for support vector machine.

1.2 Sparse Regularization

The regularization function $g(z)$ generally outputs a larger value if its input $z$ is less sparse. It is therefore also called the penalty function. Note that regularization can be use to implement hard constraints: limiting the search space to a sparse set $\mathcal{C}$ is the same as setting

$$
g(z) = \begin{cases} 
0 & : z \in \mathcal{C} \\
\infty & : z \notin \mathcal{C}
\end{cases}
$$

(1.3)

Here we list some common sparse-promoting penalty functions.

The $\ell_0$ and $\ell_1$ Norm

The $\ell_0$ norm directly indicates the number of nonzeros in a vector, so it seems straightforward to set $g(z) = \|z\|_0$ to promote sparsity. However, it would make an optimization problem combinatorial. The $\ell_1$ norm is often used as a surrogate, since it also promotes exact sparsity, yet it is convex and therefore much easier to optimize. Other choices of $\ell_p$ norm with $0 \leq p \leq 1$ also promotes sparsity (See Figure 1.1).
Figure 1.1: Unit spheres of different $\ell_p$ norms. The $\ell_1$ ball is the smallest $p$ that so that the ball is convex with pointy edges.

Capped $\ell_1$

The capped $\ell_1$ penalty is defined by

$$g(z) = \sum_{i}^{n} \max(|z_i|, \tau)$$

(1.4)

where $\tau > 0$ is a parameter that sets the max penalty. Compared to $\ell_1$ penalty, the capped $\ell_1$ penalty is a closer approximation of the $\ell_0$ norm, because larger coefficients are not penalized as much.

Log Sum

Similar to the capped $\ell_1$ penalty, Log sum is also designed to have a near constant penalty for larger coefficients. It is defined by

$$g(z) = \sum_{i}^{n} \log(1 + \frac{|z_i|}{\tau})$$

(1.5)

where $\tau > 0$ is a scaling parameter.
Smoothly Clipped Absolute Deviation (SCAD)

The smoothly clipped absolute deviation penalty is designed to be more stable in the context of feature selection by having lower bias than $\ell_1$ [27]. Regression models using SCAD have the so-called oracle property: asymptotically, they perform as well as if the correct support set is known. The SCAD penalty is defined as:

$$g(z) = \sum_{i=1}^{n} \hat{g}(z_i) \quad \text{where} \quad \hat{g}(y) = \begin{cases} \theta \cdot |y| & : |y| < \theta \\ \frac{-y^2 + 2\tau \theta |y| - \theta^2}{2(\tau - 1)} & : \theta \leq |y| < \theta \tau \\ \frac{1}{2}(\tau + 1)\theta^2 & : \theta \tau \leq |y| \end{cases} \quad (1.6)$$

where $\theta > 0$ and $\tau > 2$ are parameters.

Minimax Concave

The minimax concave penalty (MCP) is defined as:

$$g(z) = \sum_{i=1}^{n} \hat{g}(z_i) \quad \text{where} \quad \hat{g}(y) = \begin{cases} \theta \cdot |y| - \frac{1}{2\tau} y^2 & : |y| < \tau \theta \\ \frac{1}{2\tau} \theta^2 & : \tau \theta \leq |y| \end{cases} \quad (1.7)$$

MCP has the same oracle property as SCAD.
ELASTIC NET

Elastic net combines the $\ell_1$ regularization with ridge regularization:

$$g(z) = \|z\|_1 + \tau \|z\|_2^2$$

(1.8)

Elastic net can provide more stable solutions than just using $\ell_1$ (with respect to changes in the regularization strength $\lambda$) [80].

GROUPED $\ell_1/\ell_p$ NORMS

The regularization functions presented so far are all coefficient-wise separable. However, when we expect group structures in the solution, it is natural to penalize groups of variables accordingly:

$$g(z) = \sum_{g \in \mathcal{G}} w_g \|z_g\|_p$$

(1.9)

where $\mathcal{G}$ is the set of groups, and the $\ell_p$ norm can be something that does not promote sparsity, such as $\ell_2$ or $\ell_\infty$. Each group can have its own weights $w_g$. When using the group sparsity regularization, one should expect the active groups to be sparse, but the variables within a group would likely be dense.

Overlapping group sparsity can be used to implement many interesting structures. For example, consider a tree structure where a coefficient can only be selected if all its ancestors are nonzero [36]. Another example is to select groups of coefficients that are neighbors in a grid [2]. These structures can be implemented by penalizing groups of coefficient in specific ways, as illustrated in Figure 1.3.
Figure 1.3: Tree structure and contiguous pattern can be obtained by penalizing groups of coefficients in specific ways, as marked by the dotted lines. In this illustration, coefficient \{1,2,5\} are selected in (a), and \{2,3\} are selected in (b).

1.3 Smoothness

Smoothness and convexity are actually dual properties in some sense: smoothness provides an upper bound and convexity provides a lower bound.

A function \( f \) is Lipschitz continuous on \( \Omega \) for constant \( L \) if

\[
\forall x, y \in \Omega : \| f(x) - f(y) \|_* \leq L \| x - y \|
\]

(1.10)

where \( \| \cdot \|_* \) is the dual norm\(^1\).

Lipschitz continuity means that similar inputs would give similar outputs. We say \( f \) is \( L \)--smooth if \( \partial f \) is Lipschitz continuous with constant \( L \). This is useful in deriving bounds. One of the well-known bounds for \( L \)--smooth functions \( f \) is:

\[
| f(z) - f(x) - \langle \partial f(x), z - x \rangle | \leq \frac{L}{2} \| z - x \|^2
\]

(1.11)

\(^1\)The \( \ell_p \) and \( \ell_q \) norms are dual to each other if \( \frac{1}{p} + \frac{1}{q} = 1 \), and the dual of \( \ell_2 \) norm is itself.
This allows us to derive an upper bound based on observations at $x$:

$$f(z) \leq f(x) + \langle \partial f(x), z - x \rangle + \frac{L}{2} \| z - x \|^2.$$  \hfill (1.12)

### 1.4 Convexity

A function $f$ is $\alpha$-strongly convex if

$$f(x) - f(y) \leq \langle \partial f(x), x - y \rangle - \frac{\alpha}{2} \| x - y \|^2$$  \hfill (1.13)

This is true if and only if $g(x) = f(x) - \frac{\alpha}{2} \| x \|^2$ is convex (not proven here). One can interpret $\alpha$ as the curvature of $f$. Note that this implies an lower bound based on observations at $x$:

$$f(y) \geq f(x) + \langle \partial f(x), y - x \rangle + \frac{\alpha}{2} \| z - x \|^2.$$  \hfill (1.14)

### 1.5 Examples of Cost Functions for Different Applications

#### 1.5.1 Feature Selection with LASSO

The least absolute shrinkage and selection operator (LASSO) is originally developed as a method of automatic variable selection [67]. The LASSO problem is defined by:

$$\min_z \| x - Az \|_2^2 + \lambda |z|_1$$  \hfill (1.15)

where $x$ is the prediction targets, columns in $A$ are the features, and $z$ is the variable selection vector (weights of features). See Section 5.1 for more discussion.
1.5.2 Sparse Coding and Feature Extraction

In feature extraction, one is interested in finding common patterns in a dataset \( X \). This can be expressed as the following optimization problem:

\[
D = \arg \min_{D,Z} \|X - DZ\|_2^2 + g(Z) \quad (1.16)
\]

where each column in \( D \) corresponds to a feature or pattern. The collection of features, \( D \), is called a dictionary. One may also want to encode data \( x \) using an existing dictionary. This transforms the data into some kind of feature representation \( z \). This transformation is called sparse coding, and can be described by

\[
z = \arg \min_z \|x - Dz\|_2^2 + g(z) \quad (1.17)
\]

The most common regularization function used in this context is the \( \ell_1 \) norm and group sparsity. See Section 5.2 for more discussion.

1.5.3 Compressive Sensing

In compressive sensing, a signal \( x \) is compressively sensed by a sensing matrix \( A \in \mathbb{R}^{m \times n} \) with \( p \ll n \). In this context, one is interested in solving one of the following problems to recover a sparse representation of the signal:

\[
\min_z \|y - Az\|_2^2 + \lambda |z|_1 \quad \text{or} \quad \min_z \|y - Az\|_2^2 \quad \text{subject to} \quad |z|_0 < k \quad (1.18)
\]

When the sensing process involves some nonlinear transformation \( T \) in addition to the sensing matrix \( A \), we can extend the CS framework to incorporate the nonlinear-
ity. For the nonlinear extension, the corresponding problem is:

$$\min_z \|y - T(Az)\|_2^2 + \lambda |z|_1 \quad \text{or} \quad \min_z \|y - T(Az)\|_2^2 \quad \text{subject to} \quad |z|_0 < k \quad (1.19)$$

where \( T(\cdot) : \mathbb{R}^m \rightarrow \mathbb{R}^m \) is a nonlinear distortion function. More details about compressive sensing is presented in Section 5.3.

1.5.4 Classifier

The logistic regressor (LR) and the support vector machine (SVM) are two very widely used linear classifier models. They can both be kernelized to handle nonlinear data.

The optimization problems associated with these classifiers are:

(sparse LR) \[ z = \arg \min_z \frac{1}{n} \sum_i \log(1 + \exp(-y_i x_i^T z)) + g(z) \quad (1.20) \]

(sparse SVM) \[ z = \arg \min_z \frac{1}{n} \sum_i \max(0, 1 - b_i x_i^T z)^2 + g(z) \quad (1.21) \]

were \((x_i, y_i) \in (\mathbb{R}^n, \mathbb{R})\) are the training samples, and \(z\) is the learned parameters of the classifier. Section 5.4 presents a cancer subtype classification problem that can be modeled by sparse LR.

1.6 Summary and Remarks

Sparse regularization offers two benefits: (1) it prevents over-fitting by controlling the complexity of models, and (2) it makes a model more interpretable\(^2\) by isolating important variables. In practice, sparse modeling often removes noise and reveals

\(^2\)Sparse representation is known to be useful in machines learning, and it is easier for humans to understand a model if there are fewer factors.
structure in data.

There are many forms of sparse regularization, including $\ell_p$ norm and capped $ell_p$ norm with $p < 1$, MCP, SCAD, group sparsity, and hierarchical sparsity. There are also many choices of fitness functions such as squared loss and hinge loss. In this thesis we only consider fitness functions that are smooth and differentiable.

Currently, the fastest sparse solvers only works for separable regularization, and the general solvers do not account for the sparsity of solutions explicitly. We will address this issue in Section 4 by proposing a general method that explicitly leverages the fact that the solution is sparse.
“Sometimes an inaccurate formula is a handy way to move you in the right direction if it offers simplicity.”
— Scott Adams

Gradient descent has been expanded greatly since it was invented by Cauchy in 1847 [18]. While being extremely simple in concept, it is one of the most robust and efficient tools to solving optimization problems. In this section, we use a general framework to describe some of the most important variations of gradient descent methods.
2.1 **Gradient Descent**

Gradient descent is simple: at a point \( x^k \), we simply take a small step towards the direction of steepest descent. However, there is an alternative interpretation that generalizes to various gradient methods — gradient descend steps are effectively minimizing an approximation of the cost function at a given point \( x^k \). Many seemingly different descent methods are often just using different types of approximation functions under the same framework.

Let \( Q(x) \) be the objective function we want to minimize. Consider the following quadratic approximation of \( Q \) at a given point \( y \):

\[
\tilde{Q}_{y,L}(x) = Q(y) + \partial Q(y)^T (x - y) + \frac{L}{2} \| x - y \|^2
\]  

(2.1)

If \( Q : \mathbb{R}^n \to \mathbb{R} \) is a continuously differentiable function with Lipschitz continuous gradient with constant \( L_Q \), then for any \( L > L_Q \) we have:

\[
\tilde{Q}_{y,L}(x) \geq Q(x)
\]  

(2.2)

which means \( \tilde{Q} \) is an upperbound of \( Q \).

Gradient descent works by minimizing this upper bound. With some algebraic manipulation, the minimizer of \( \tilde{Q}_y(x) \) at a point \( y \) can be written as

\[
\arg\min_{x \in \mathbb{R}^n} \tilde{Q}_{y,L}(x) = \| x - (y - \frac{1}{L} \partial Q(y)) \|^2
\]  

(2.3)

which clearly shows that the minimizer is \( y - \frac{1}{L} \partial Q(y) \). This leads to the gradient
descent method where the steps are defined by:

$$z^{k+1} = \arg \min_{z \in \mathbb{R}^n} \hat{Q}_{z_k, L}(z) = z^k - \frac{1}{L} \partial Q(z^k)$$ (2.4)

For convex $Q$, the sequence generated by (2.4) satisfies:

$$Q(z^k) - Q(z^*) \leq \frac{c L Q}{2k} \|z^0 - z^*\|_2^2$$ (2.5)

where $z^*$ is a minimizer of $Q$.

### 2.2 Proximal Gradient Descent (PG)

Proximal gradient descent is designed to leverage the fact that the cost function $Q$ is a sum of two parts, namely the fitness function $f$ and the regularization term $g$.

There are two motivations for doing this: First, by separating these terms, we can get a better upperbound than (2.1). Secondly, this allows us to deal with non-differentiable regularization terms.

Consider the following upperbound of $Q = f + g$ at a given point $y$:

$$\hat{Q}_{y, L}(x) \equiv f(y) + \partial f(y)^T(x - y) + \frac{L}{2} \|x - y\|_2^2 + g(x)$$ (2.6)

with some $L \geq L_f$. This is a better bound than (2.1), because we are only approximating $f$ while using $g$ as it is. The steps in proximal methods minimizes this new upper bound. Similar to 2.3, the minimizer of $\hat{Q}(x)$ can be written as

$$\arg \min_{x \in \mathbb{R}^n} \hat{Q}_{y, L}(x) = \arg \min_{x \in \mathbb{R}^n} g(x) + \frac{L}{2} \|x - (y - \frac{1}{L} \partial f(y))\|_2^2$$ (2.7)
One can view the minimization problem in (2.7) as minimizing $g$ within some proximity within the gradient step based on $f$. To be more precise, we introduce the proximal operator, which is defined by:

$$\text{prox}_h(v) \equiv \arg \min_{x \in \mathbb{R}^n} h(x) + \frac{1}{2} \|x - v\|_2^2$$  \hspace{1cm} (2.8)

The notations in 2.7 can be reorganized using the proximal operator, and now we have the proximal gradient descent step:

$$z^{k+1} = PGS_L(z^k)$$  \hspace{1cm} (2.9)

$$= \arg \min_{z \in \mathbb{R}^n} Q_{z^k, L}(z)$$  \hspace{1cm} (2.10)

$$= \arg \min_{z \in \mathbb{R}^n} g(z) + \frac{L}{2} \|z - (z^k - \frac{1}{L} \partial f(z^k))\|_2^2$$  \hspace{1cm} (2.11)

$$= \text{prox}_{\frac{1}{L} g} (z^k - \frac{1}{L} \partial f(z^k))$$  \hspace{1cm} (2.12)

Note that the regularization function $g$ is only accessed through the proximal operator. This means we can use any $g$ as long as we can evaluate the corresponding proximal operator.

\textbf{Algorithm 1} Proximal Gradient Step (PGS$_L$)

\begin{itemize}
  \item \textbf{input} $x$
  \item $y := \text{prox}_{\frac{1}{L} g} (x - \frac{1}{L} \partial f(x))$
  \item \textbf{output} $y$
\end{itemize}
2.2.1 Evaluating the Proximal Operator

Proximal operators for certain regularization functions can be computed very efficiently. For $\ell_1$ regularization ($g(z) = \lambda \|z\|_1$), the proximal operator is soft-threshold:

$$[\text{soft-threshold}_\lambda(v)]_i = \text{sign}(v_i)(|v_i| - \lambda)_+$$

When $g$ is used to implement a constraint, the proximal operator are projections onto the feasible solution space $\mathcal{C}$. The proximal operator for the $\ell_0$ constraint (i.e., $\mathcal{C} = \{x : \|x\|_0 < K\}$) is simply the hard-threshold function:

$$[\text{hard-threshold}_k(v)]_i = \begin{cases} v_i & : \text{if } v_i \text{ is one of the largest } K \text{ coefficients in magnitude} \\ 0 & : \text{otherwise} \end{cases}$$

which is a projection onto the $\ell_0$ ball with norm $K$. Similarly, for $\ell_1$ constraint the proximal operator is a projection onto the $\ell_1$ ball. In these cases the proximal gradient descent generalizes to projected gradient descent. PG for the $\ell_0$ constraint is a noteworthy special case, because under some specific cases these algorithms can still find the global optimal even when the problem is non-convex (For example, see iterative hard-thresholding (IHT) [10] and gradient descent with sparsification [31].)

For other regularization functions such as overlapping group sparsity, the proximal operator may involve solving a minimization problem. See [56] for a more complete discussion.

The proximal operators for sparsity inducing regularization functions tend to shrink its inputs, therefore they are sometimes called shrinkage operators. When used in the context of sparse optimization, the proximal gradient descent is often referred to as
the iterative shrinkage-thresholding algorithm (ISTA).

2.2.2 Selecting Step Size

The step size $\frac{1}{L}$ in PGS is determined by our choice of $L$. In the previous section we have assumed that $L_f$ is known and we can select any $L \geq L_f$. When $L_f$ is unknown or hard to compute, one simple strategy is to use backtracking to find an appropriate $L$ efficiently [5], as shown in Algorithm 2. The parameter $\eta$ controls how fast we increase $L$ when the estimated upperbound $\hat{Q}$ based on $L$ is violated. Typically we set $\eta = 2$. For simplicity, we will use (PGS) to denote the proximal gradient step with constant stepsize $\frac{1}{L}$ in the rest of this document.

<table>
<thead>
<tr>
<th>Algorithm 2 Proximal Gradient Step (PGS) with Backtracking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong> $x, L$, parameter $\eta$</td>
</tr>
<tr>
<td>repeat</td>
</tr>
<tr>
<td>$y := \text{PGS}_L(x)$</td>
</tr>
<tr>
<td>$L := \eta L$</td>
</tr>
<tr>
<td>until $Q(y) \leq \hat{Q}_{x,L}(y)$</td>
</tr>
<tr>
<td><strong>output</strong> $y, \frac{1}{\eta}L$</td>
</tr>
</tbody>
</table>

2.2.3 Accelerated Proximal Gradient Descent (APG)

An acceleration method due to Beck and Teboulle [4] evaluates the gradient step at an extrapolated point $y^k$ rather than $z^k$:

\begin{align}
    z^{k+1} &= \text{PGS}(y^k) \\
    t^{k+1} &= (1 + \sqrt{1 + (2t^k)^2}) / 2 \\
    y^{k+1} &= z^{k+1} + \frac{t^{k-1} - 1}{t^k}(z^{k+1} - z^k)
\end{align}

(2.13) \hspace{1cm} (2.14) \hspace{1cm} (2.15)
For convex $Q$, the sequence generated by (2.13) satisfies:

$$Q(z^k) - Q(z^*) \leq \frac{cL_f}{(k+1)^2} \|z^0 - z^*\|_2^2$$

(2.16)

The method of (2.13) through (2.15) is called the fast iterative shrinkage thresholding algorithm (FISTA).

The earliest accelerated gradient descent was due to Nesterov [49], and there has been more follow up research on this topic [50, 51, 53, 53, 69]. These accelerated schemes achieves optimal convergence rate among all first-order methods. While being simple algorithmically, an intuitive understanding of these accelerated schemes has always been elusive. It is only until very recently that a connections to ODE was shown by Su et al [65].

FISTA is not a monotone algorithm, as $Q(z^k)$ may now be larger than $Q(z^{k-1})$. However, Beck and Teboulle provided the following simple modification to (2.13), which guarantees that the objective function is non-increasing over iterations:

$$s^{k+1} = \text{PGS}(y^k)$$

(2.17)

$$z^{k+1} = \begin{cases} s^k : Q(s^{k+1}) < Q(z^k) \\ z^k : \text{otherwise} \end{cases}$$

(2.18)

It is worth noting that the momentum stops whenever we set $z^{k+1} = z^k$, and the extrapolation in the next iteration is temporarily disabled. This has a very similar effect to the adaptive restart scheme [55].
2.2.4 Accelerated Proximal Gradient for Non-convex Optimization

The convergence of FISTA for non-convex programming has been recently studied by Li and Lin [39], who propose a further extension that guarantees convergence to a critical point. In this method, monotonic accelerated proximal gradient descent (mAP), a non-extrapolated step is evaluated and selected if it descends more than the extrapolated step:

\[ v^{k+1} = \text{PGS}(y^k) \]  \hspace{1cm} (2.19)
\[ m^{k+1} = \text{PGS}(x^k) \]  \hspace{1cm} (2.20)
\[ z^{k+1} = \begin{cases} 
  v^{k+1} & : f(v^{k+1}) \leq f(m^{k+1}) \\
  m^{k+1} & : \text{otherwise}
\end{cases} \]  \hspace{1cm} (2.21)
\[ t^{k+1} = (1 + \sqrt{1 + (2t^k)^2})/2 \]  \hspace{1cm} (2.22)
\[ y^{k+1} = z^{k+1} + \frac{t^{k+1}-1}{t^k}(z^{k+1} - z^k) \]  \hspace{1cm} (2.23)

Evaluating both steps roughly doubles the computation cost of each iteration. However, it can be avoided by computing the non-extrapolated step only when the extrapolated step fails to descend sufficiently. This slightly changes the mAP algorithm, and for the purpose of this thesis we will not discuss those extensions as they do not improve convergence rates.

2.2.5 Generalization to Second Order Methods

In the proximal gradient methods present so far, we have been using an approximation based on the smoothness of \( f \). That is, we approximate \( f \) by a linear model plus a quadratic residual. However, the quadratic term could be improved.
Let us rewrite the upperbound in (2.6) as:

\[ \hat{Q}_y(x) \equiv f(y) + \partial f(y)^T (x - y) + D(x, y) + g(x) \]  

where \( D(x, y) = \frac{1}{2} \| x - y \|^2 \) is the quadratic term. The upperbound in (2.24) generalizes to Newton’s method if \( D(x, y) = \frac{1}{2} \| x - y \|_H \) where \( H \) is the Hessian matrix with respect to \( f \) (The variable metric \( \| w \|_A \) is defined by \( \| w \|_A = w^T A w \)). However, it is often more efficient to use quasi-Newton methods that efficiently approximates \( H \). Notable examples includes DFP [30], BFGS [63] and SR1 [14]. Another example is Barzilai-Borwein stepsize, which approximates \( H \) with a scalar (the step size).

A gradient step with respect to the (approximate) Hessian matrix in the upperbound becomes the following:

\[ z^{k+1} = \arg \min_{z \in \mathbb{R}^n} \hat{Q}_{z^k}(z) \]
\[ = \text{prox}^H_y \left( z^k - H^{-1} \partial f(z^k) \right) \]

where the proximal operator with respect to \( H \) is defined by

\[ \text{prox}^H_v \equiv \arg \min_{x \in \mathbb{R}^n} h(x) + \frac{1}{2} \| x - v \|_H \]

Note that this is a generalization of the proximal operator presented in (2.8). Evaluating the generalized proximal operator can be quite difficult, and requires other approximation methods. There is some recent work by Becker and Fadili on solving these quasi-Newton proximal methods efficiently [6].

25
2.2.6 Generalization to Mirror Descent

In the previous section, we discussed how PG can be generalized to second order methods by setting $D(x, y) = \frac{1}{2} \|x - y\|_H$ in (2.24), and noted that evaluating the generalized proximal operator can become difficult. Mirror descent takes the opposite direction — it selects $D(x, y)$ tailored for specific regularization or constraints $g$ so that the proximal operator becomes easy to evaluate\(^1\).

In mirror descent, $D(x, y)$ is set to be the Bregman divergence with respect to some function $\psi$:

$$D(x, y) := B_\psi(x, y) \equiv \psi(x) - \psi(y) - \langle x - y, \psi'(y) \rangle$$ (2.28)

Note that setting $\psi(x) = \|x\|_2^2$ leads us back to the proximal gradient method where $D(x, y) = \|x - y\|_2^2$. However, it also generalizes to many other useful distances, such as KL-divergence when $\psi(p) = \sum_i \{p_i \log p_i - p_i\}$. Mirror descent has been shown to work very well with the simplex constraint \([3, 7]\), which is useful for optimization problems where the coefficient vector represents a probability distribution\(^2\).

2.3 Coordinate Descent (CD)

The general form of coordinate descent updates the coefficients sequentially, as shown in Algorithm 3. While they are not designed for sparse problems initially, CD works very well in finding sparse solutions.

Coordinate descent is used only when $g$ can be expressed as $g(z) = \sum_i^n g(z_i)$ (i.e.,

\(^1\)The quadratic term is chosen so that the convergence guarantees can be retained.
\(^2\)In this case, $z^*$ is a positive vector that sums to 1, which can be enforced by a simplex constraint.
Algorithm 3 Coordinate Descent

Initialize $z^0 \in \mathbb{R}^n$

repeat
   Select $i \in \{1, 2, \ldots, n\}$
   $z_i := \text{prox}_{\frac{1}{L}g}(z_i - \frac{1}{L}\partial f(z)_i)$

until termination condition is met

output $z$

$g$ is a separable function, such as the $\ell_1$ norm. Since the coordinate update only involves scalar optimizations, sometimes the update can be solved in closed form. In these cases, the coordinate update becomes:

$$[z^{k+1}]_{i_k} := \arg\min_{z_{i_k}} f(z) + g(z) \quad \text{subject to} \quad z_j = [z^k]_j \quad \forall j \neq i_k \quad (2.29)$$

There are several strategies for selecting the coordinate $i_k$ at each iteration. Cyclic coordinate descent cycles through the coordinates in order. Random coordinate descent chooses $i_k$ at random, and it actually has better convergence guarantees over the cyclic version [52]. For sparse optimization problems, choosing $i_k$ in a greedy fashion (Gauss–Southwell rule) has also been shown to work well\(^3\)[54]. Note that the greedy selection process usually involves some additional cost compared to cyclic and random coordinate descent. For example, if we select $i_k$ based on the largest gradient $|\partial f(z^k)|_{i_k}$, then the full gradient needs to be computed or maintained.

Block coordinate descent (BCD) is a generalization of CD that considers a group of coefficients at the same time. BCD can handle regularization functions that are separable by non-overlapping groups (i.e. $g(z) = \sum_{\Omega_i \in \Omega} g(z_{\Omega_i})$ where $\Omega_i$ is a partition of $\Omega$). One of such example is the non-overlapping group lasso. Many other methods

\(^3\)Greedy CD is faster without concerning scalability. Things are much more complicated when parallelization and data distribution needs to be considered.
can be viewed as special cases of BCD. For example, the alternating direction method of multipliers (ADMM) can be viewed as BCD that alternates between two variable blocks.

2.4 Stochastic Gradient Descent (SGD)

The computational effort in (proximal) gradient descent methods is mostly spent on computing the gradient $\partial f$ (and proximal operator $\text{prox}_g$ for the more complex $g$). Stochastic gradient descent methods are designed to greatly reduce the gradient computation cost for machine learning applications.

In the context of supervised learning, the fitness function $f$ is the sum of loss for individual samples:

$$\min_z \left\{ f(z) \equiv \frac{1}{N} \sum_{i} \text{loss}(\mathcal{M}_z(x_i), y_i) \right\}$$

where the training set consists of $N$ tuples of training samples $(x_i, y_i)$. The gradient of $f$ is therefore the average of gradients derived from individual samples:

$$\partial f(z) = \frac{1}{N} \sum_{i} \partial \text{loss}(\mathcal{M}_z(x_i), y_i)$$

The cost of computing $\partial f$ scales linearly with the number of samples, and it becomes a great burden with a very large training set.

Stochastic gradient descent computes $\partial f$ approximately using just a subset of samples at random. It is easy to see that the average of a subset would be close to the average of the entire training set. In fact, it has been shown that stochastic gradient descent can converge even by just evaluating a single sample in each step, provided
that the step size is small enough and converges to 0.

The stochastic nature means we can no longer use methods like backtracking to select stepsize. The choice of stepsize (or learning rate in the context of SGD) becomes a difficult issue. Since SGD is pretty much the only way to deal with huge amount of training data (e.g., training neural nets), there has been a lot of work devoted to address the learning rate problem. Some of the notable examples includes the momentum SGD [62], RMSprop [22, 33], Nesterov’s accelerated gradient method [8], adaGrad [25], adaDelta [79] and ADAM [37].

2.5 Summary and Remarks

There are many variations of gradient descent developed from different lines of work. However, many of them fit into the general framework that explains each gradient step as minimizing an local quadratic approximation of the objective function. Proximal gradient descent improves the approximation by splitting the objective function into two parts, and solves the resulting minimizing problem with proximal operators. Second order methods further refine the approximation using the Hessian (or approximations of the Hessian), which result in proximal operators that are more difficult to evaluate. Mirror descent modifies the approximation in a way that simplifies the evaluation of the proximal operator. Coordinate descent is a special case where the changes are restricted to a single dimension in the solution space in each iteration.

In practice, accelerated first order methods and coordinate descent works better for solving sparse optimization problems. In particular, coordinate descent is considered one of the most competitive methods for solving $\ell_1$ regularized problems, and FISTA (and its variations) is considered the fastest for solving problems with structured sparsity. The second order methods may have better convergence guarantees,
but the overhead of maintaining the Hessian (or an approximation to it) and the more expensive proximal operator outweighs the benefit. None of the gradient methods take the sparse assumption into account explicitly, so there are much room for improvements when we know the solution is sparse\(^4\).

\(^4\)Greedy CD is the only one specifically designed for finding sparse solutions, but it lacks scalability.
Homotopy and Working Set Methods

Homotopy and working set methods explicitly leverage the sparsity of solutions for better efficiency. These methods build up a support set\(^1\) gradually until the optimal support set is reached. They differ from the gradient methods in that the gradient methods tend to start with very dense estimations, and gradually sparsify it through iterative shrinkage.

Homotopy methods start out with strong regularization \(\lambda^0 > \lambda\) with trivial solution at 0, then solve problems by decreasing \(\lambda^k\) until the desired regularization is reached (\(\lambda^K = \lambda\)). This approach requires: (1) a solver that benefits from warm start, and (2) a strategy to select the intermediate regularization parameters \(\{\lambda^0, ..., \lambda^K = \lambda\}\).

\(^1\)Support set means the set of nonzero coefficients.
Working set methods also start out with 0, but they solve subproblems with expanding the support set until the optimality condition is reached. This approach requires (1) an innerloop solver, (2) an expansion rule and (3) an optimality check. The advantage of working set methods comes from the reduced problem size in innerloops (proportional to size of support set), and that the innerloop can often be solved in close form.

3.1 Least Angle Regression (LARS)

The Least Angle Regression (LARS) algorithm is a homotopy method for solving the LASSO problem:

\[(LASSO) \quad \min_z \left\{ \frac{1}{2} \|y - Xz\|_2^2 + \lambda \|z\|_1 \right\}\]

where \(y \in \mathbb{R}^N\) and \(X \in \mathbb{R}^{N \times n}\). Let us denote the solution for a given \(\lambda\) as \(z(\lambda)\), and consider the solution function \(S : \lambda \rightarrow z(\lambda)\). This function \(S\) happens to be piecewise linear, as shown in Figure 3.1.

![Figure 3.1: The solution \(z(\lambda)\) at different \(\lambda\). Note that the transition is piecewise linear. The turning points are indicated with vertical dotted lines.](image)

32
Based on the piecewise linearity, one can compute the directions at \( z(\lambda^k) \), and follow that direction linearly until a turning point \( z(\lambda^{k+1}) \) is reached. Specifically, we can set \( \lambda^0 = \|X^Ty\|_\infty \), which gives the trivial solution \( z(\lambda^0) = 0 \). Then, we can express the solution function \( S \) around \( z(\lambda^k) \) in closed form:

\[
[z(\lambda')]_\Omega = (X^T_{\Omega} X_{\Omega})^{-1} (X^T_{\Omega} y - \lambda' \cdot \text{sign}(z(\lambda^k))_{\Omega}) \tag{3.1}
\]

\[
[z(\lambda')]_{\Omega^c} = 0 \tag{3.2}
\]

where \( \Omega \) is the support set of \( z(\lambda^k) \). This linear function can be used to compute \( z(\lambda') \) for decreasing \( \lambda' \) until a turning point is reached. The conditions of reaching a turning point are:

1. There exists \( i \in \Omega^c \) such that \( \|X^T_i (y - X z(\lambda^k))\|_1 = \lambda' \). In this case, add \( i \) to \( \Omega \) and continue.

2. There exists \( i \in \Omega \) such that \( z(\lambda')_i = 0 \). In this case, remove \( i \) from \( \Omega \) and continue.

The complexity of LARS depends on the number of turning points before reaching the target \( \lambda \). At iteration \( k \), LARS computes the exact solution for the \( k \)-th turning point (for the corresponding \( \lambda^k \)). Note that there will be at least \( K \) turning points for a solution with \( K \) nonzero coefficients.

### 3.2 Feature-Sign Search (FSS)

The Feature-Sign Search algorithm is a working set method for solving LASSO [38]. In each iteration, one coefficient is added to the working set. Then, we estimate the signs of the coefficients in the working set, and solve the subproblem in closed form.
If the estimated signs disagree with the closed-form solution, we can perform a line search to fix it. This process is repeated until the optimality condition is met, as shown in Algorithm 4:

**Algorithm 4 Feature-Sign Search**

Initialize $z^0 = 0$, $\Omega = \emptyset$, $k = 0$

repeat

- $i = \arg \max_i \| X_i^T (y - X z^k) \|$
- $\Omega := \Omega \cup i$
- $\theta := \text{sign}(z^k)$
- $[\theta]_i := \text{sign}(X_i^T (y - X z^k))$
- $\hat{z} := 0$
- $[\hat{z}]_\Omega := (X_{\Omega}^T X_\Omega)^{-1} (X_{\Omega}^T y - \lambda \cdot \theta_\Omega)$

Perform line search between $z^k$ and $\hat{z}$, and set $z^{k+1}$ to the point that gives lowest objective value.

- $\Omega := \{ j : [z^{k+1}]_j \neq 0 \}$
- $k := k + 1$

until termination condition is met

output $z^k$

The FSS algorithm is very similar to LARS: they both add coefficients with the largest gradient to the support set, and they both use closed form least-squares to estimate coefficients within the support set. The main difference is that FSS uses the same $\lambda$ over iterations, whereas the use of larger $\lambda$ in LARS can slow down the growth of coefficients. Assuming that we are only interested in finding the solution with respect to $\lambda$ and not any of the intermediate solutions for $\lambda^k$, FSS is typically faster than LARS.

### 3.3 Proximal Gradient Homotopy (PGH)

Proximal gradient homotopy method [75] is yet another solver designed for the LASSO problem in (1.15). Similar to LARS, PGH solves a sequence of problems with decreas-
ing $\lambda$. A sketch of the method is shown in Algorithm 5.

**Algorithm 5** Proximal Gradient Homotopy (PGH)

<table>
<thead>
<tr>
<th>Algorithm 5 Proximal Gradient Homotopy (PGH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize $z = 0 \in \mathbb{R}^n$, $\Omega = {}$</td>
</tr>
<tr>
<td>for $k = 0, 2, 3, ..., \log_\eta(\lambda/\lambda^0)$ do</td>
</tr>
<tr>
<td>$z^k = \arg\min_z |x - Az| + \lambda^k |z|_1$ // solved with PG</td>
</tr>
<tr>
<td>$\lambda^{k+1} = \eta \cdot \lambda^k$</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>output $z$</td>
</tr>
</tbody>
</table>

PGH is faster than directly solving (1.15) using PG for two reasons$^2$: (1) Convergence of PG is faster in regions local to the solution. (2) Solutions for $\lambda$ and $\eta\lambda$ should be reasonably close. Intuitively, PGH is always chasing an intermediate solution that is not far from its current state, and once it gets close enough, we update $\lambda^k$ to make the intermediate problem closer to the master problem.

One problem with PGH is that the bias from using larger lambdas hurts their estimation of large coefficients in early iterations. As discussed in Section 3.2, this is similar to the reason why LARS is slower than FSS in practice.

3.4 **Matching Pursuit Lasso (MPL)**

Matching pursuit LASSO [66] is another working set method designed to solve the LASSO problem in (1.15). It is similar to FSS, with two distinctions:

1. **Support Set Expansion Rule**

   In each iteration, the $K$ coefficients with largest gradients are added to the support set. The expansion size $K$ is determined upfront by counting the number of coefficients with gradients larger than $\alpha \|A^T z\|_\infty$, where $\alpha$ is a parameter.

$^2$Instead of PG, one can also apply the same principle to coordinate descent by solving the subproblems with CD.
2. Innerloop Solver

Similar to FSS, the innerloop of MPL is a least squares problem with $\ell_1$ penalty.

However, in MPL it is solved by proximal gradient descent.

Having the flexibility to add more than one coefficient to the support set in each iteration turns out to be very important. Note that FSS needs at least $\|z^*\|_0$ iterations just to build up the support set, which can be inefficient, especially for larger problems. On the other hand, the support set expansion rule in MKL involves a new parameter $\alpha$, which can result in worse performance than FSS if not tuned correctly.

3.5 Orthogonal Matching Pursuit (OMP)

The Orthogonal Matching Pursuit (OMP) [57] algorithm is a working set method designed to solve the $\ell_0$ constrained least squares problem:

$$
\min_{z} \left\{ \frac{1}{2} \|x - Az\|_2^2 \right\} \quad \text{subject to} \quad \|z\|_0 < \lambda
$$

(3.3)

where the parameter $\lambda$ is a positive integer indicating how many nonzero coefficients we want in the solution.

OMP builds up the working set by adding one coefficient at a time. The innerloop is a small least squares problem concerning coefficients in the working set. This process is repeated for exactly $\lambda$ times, as shown in Algorithm 6:

While this $\ell_0$ constrained problem is not convex, OMP can still find the correct global optimal solution with some assumptions on the matrix $A$ with respect to parameter $\lambda$ [68].
Algorithm 6 Orthogonal Matching Pursuit

Initialize $z = 0 \in \mathbb{R}^n$, $\Omega = \{\}$

for $k = 1, 2, 3, ..., \lambda$ do
  $i = \arg \max_i \|A_i^T(x - A)\|$  
  $\Omega := \Omega \cup i$  
  $z_\Omega := (A_{\Omega}^T A_\Omega)^{-1} (A_{\Omega}^T x)$
end for

output $z$

3.6 Summary and Remarks

Homotopy and working set methods explicitly take into account that the solution is sparse. Homotopy methods use warm start to efficiently solve a sequence of subproblems with decreasing $\lambda^k$. They are very efficient in finding solutions for multiple levels of regularization strength, which could be useful for cross validating $\lambda$. However, the interim $\lambda^k$’s throughout the process could create unnecessary bias towards 0 when we are only interested in the original $\lambda$.

Working set methods try to build up the correct support set by solving a sequence of low-dimensional subproblems. These methods work up to the correct support set rather conservatively, adding only few coefficients in each iteration. Therefore, finding the correct support set reliably and efficiently is crucial.
Auxiliary Sparse Homotopy (ASH)

“Not all those who wander are lost.”
— J. R. R. Tolkien

We present a way to modify gradient methods so that they try to stay sparse as long as they are making progress. While this may slowdown the descending speed locally, it is actually taking a shortcut by traveling in a refined solution space.

4.1 Intuition

Proximal methods such as FISTA are among the most efficient solvers for sparse problems, especially at large scales. However, they generally spend a lot of effort in
updating coefficients that eventually shrinks to zero. Suppose we start from $z^0 = 0$, for most coefficients we will have

$$\sum_{k=0}^{K} (z^{k+1} - z^k)_i = 0$$

(4.1)

when the algorithm terminates after $K$ iterations. These null updates have two drawbacks — not only do they waste computation effort, these nonzero intermediate states may actually interfere with useful updates. As illustrated earlier in Figure 1, updates that cause the support set to expand are mostly null updates when the solution is sparse.

Our core idea is to modify proximal gradient steps (PGS) by encouraging them to traverse through sparse interim solutions. First, it can potentially speed up convergence by avoiding null updates. Secondly, it is computationally efficient to operate on a small support. The modification should be done carefully as to maintain the convergence properties of the original algorithms.

4.2 A Working Set Approach

We can learn from existing working set methods and try to build up support set for gradient descent methods.

By keeping a small support set, we can focus on getting good approximations for a small number of coefficients that are identified to be important. During gradient steps, we can gradually add coefficients to the support set only when they make a big enough difference. Specifically, we keep coefficient $i$ in the support set when $|\text{PGS}(z^k)_i|$ is larger than one of the following values:

1. Previous Smallest: $\rho \cdot \tau^{k-1}$
2. **Current Smallest:** \( \tau_1 = \min_{j \in \Omega_{z^k}} (|\text{PGS}(z^k)_j|) \)

3. **Large Update:** \( \tau_2 = \frac{1}{2}(\|\delta_{z^k}\|_\infty + \|\delta\|_\infty) \) where \( \delta = \text{PGS}(z^k) - z^k \).

To summarize, \( \tau^{k-1} \) remembers the smallest nonzero coefficient in past iterations, \( \tau_1 \) is the smallest coefficient in the previous support set, and \( \tau_2 \) captures the magnitude of updates in PGS\(^1\). We still perform PGS in each iteration, but any coefficients less than \( \tau^k = \min(\{\rho \cdot \tau^{k-1}, \tau_1, \tau_2\}) \) is set to zero.

This is different from existing working set methods in that the support set is expanded in a dynamic way.

### 4.3 A Homotopy Approach

Another idea is to artificially make it more difficult for coefficients to enter the support set, and only lower the bar when it is necessary for making progress. This allows the gradient steps to focus on the important coefficients first, thus mitigating the interference between coefficients. Consider adding a thresholding constraint to the original problem in (1.1). That is, the coefficients of the solution are either 0 or greater than \( \tau^* \). The new problem is:

\[
\min_x \{Q(x)\} \quad \text{subject to} \quad |z_i| \geq \tau \quad \forall |z_i| > 0
\]  

When \( \tau = 0 \), this clearly has the same minimizer as (1.1).

However, if we solve (4.2) using the homotopy approach with respect to \( \tau \), then we would have sparse interim solutions that converges towards \( z^* \). Specifically, we

\(^1\)In the third rule, what we really want is to keep coefficient \( i \) if \( |\text{PGS}(z^k) - z^k_i| \geq \tau_2 \). However, if \( z^k_i \) is nonzero than coefficient \( i \) will be added by the second rule anyway, so \( \tau_2 \) only affects cases where \( z^k_i = 0 \).
can start by solving 4.2 with a large $\tau$, and then gradually reduces $\tau_i$ and solve the corresponding problems using warm start until $\tau$ reaches 0.

It is clear that this strategy would yield a solution to (1.1). However, the actual performance is determined by the sequence of $\tau$ and the precision we solve each sub-problem. The solution to each subproblem is approximated by exactly one (proximal) gradient step with respect to $Q$, followed by a thresholding function based on $\tau$. One simple way is to use a decreasing geometric sequence for $\tau$:

$$
\tau^k = \rho \cdot \tau^{k-1} \quad (4.3)
$$

where $\rho \in [0, 1]$ is a parameter that controls how fast $\tau$ decreases.

A more adaptive strategy is to consider how small $\tau$ needs to be in order to make sufficient progress. For example, we can set $\tau$ in a way so that the largest updates will always be included:

$$
\tau_1 = \min_{i \in \Omega(z^k)} |\text{PGS}(z^k)_i| \quad (4.4)
$$

$$
\tau_2 = \frac{1}{2}(\| \delta_{\Omega(z^k)} \|_{\infty} + \| \delta \|_{\infty}) \quad \text{where} \quad \delta = \text{PGS}(z^k) - z^k \quad (4.5)
$$

$$
\tau^k = \min(\{\rho \cdot \tau^{k-1}, \tau_1, \tau_2\}) \quad (4.6)
$$

In (4.6), $\tau_1$ is the smallest value already inside the support set, and $\tau_2$ is the smallest coefficient we want to add into the support set. By setting the penalization threshold to the minimum of these values, we are essentially expanding the support set in each iteration. Empirically, the adaptive $\tau$ often outperforms the simple rule in (4.3), and it is much less sensitive to the parameter $\rho$. In both settings, we can initialize $\tau^0$ as $\xi \cdot \max_i |[\text{PGS}(z^0) - z^0]|$ with $\xi \in [0, 1]$. If $\xi = 1$, then we are keeping only the
largest coefficient.

4.4 The Auxiliary Sparse Homotopy Method (ASH)

The two approaches presented in the previous section leads to the same method, in which a standard proximal gradient step (PGS) is followed by a thresholding function:

\[ [\text{threshold}_r(y)]_i \equiv \begin{cases} y_i & |y_i| \geq \tau \\ 0 & |y_i| < \tau \end{cases} \]  

(4.7)

We call this new step shown in Algorithm 7 as the sparse proximal gradient step (SPGS).

**Algorithm 7 Sparse Proximal Gradient Step (SPGS)**

**input** $x, L, \tau$, parameter $\eta$

**repeat**
- update $\tau$ using (4.6),
- $x^+ := \text{PGS}_L(x)$
- $y := \text{threshold}_r(x^+)$
- $L := \eta L$

**until** $\min(Q(y), Q(x^+)) \leq \hat{Q}_{x,L}(x^+)$

**output** $y, \frac{1}{\eta} L$

We can now replace PGS with SPGS in gradient descent methods such as ISTA, FISTA or mAP, as shown in Algorithm 8. Note that the initial $z^0$ needs to be sparse — if $z^0$ is dense, then SPGS behaves exactly the same as PGS. We call this the auxiliary sparse homotopy method, or ASH. We will use this acronym loosely for both the general approach and the resulting algorithms.

---

Note that this thresholding function corresponds to the auxiliary constraint that we imposed onto the problem to encourage sparse interim solutions, and it is not the hard-thresholding function (proximal operator) that comes from having a $\ell_0$ related $g$.
**Algorithm 8** Auxiliary Sparse Homotopy for Gradient Descent Methods (ASH)

Initialize $z^0 = 0$.

for $k = 1, 2, 3, \ldots$ do

update $z^k$ using rule sets from ISTA, FISTA or mAP,

but replace PGS with SPGS.

end for

output $z^k$

It is trivial to see that ASH is correct from the homotopy view: SPGS efficiently becomes PGS over iterations since the threshold $\tau$ decreases exponentially. The idea is that $z^k$ is already close to the solution by the time SPGS becomes PGS.

**Lemma 1** (Correctness). For $k \geq \log_{\rho} \frac{2\epsilon}{L\sqrt{n\tau^0}}$, we have

$$|Q(\text{SPGS}(z^k)) - Q(\text{PGS}(z^k))| \leq \epsilon$$

**Proof.** This is trivial from the fact that $\tau^k$ converges to 0.

\begin{align}
|Q(\text{SPGS}(z^k)) - Q(\text{PGS}(z^k))| &\leq \frac{L}{2}\|\text{SPGS}(z^k) - \text{PGS}(z^k)\| \\
&\leq \frac{L}{2} \sqrt{\sum_i (\tau^k)^2} \\
&\leq \frac{L}{2} \sqrt{n\rho^k \tau^0}
\end{align}

where $L$ is the Lipschitz constant of $f$. Note that $\frac{L}{2} \sqrt{n\rho^k \tau^0} \leq \epsilon$ if $k \geq \log_{\rho} \frac{2\epsilon}{L\sqrt{n\tau^0}}$.

**4.4.1 ASH for Separable Regularization**

We can avoid computing the full PGS in Algorithm 7. If we could predict the support set after pruning, then we can leverage the sparsity of $z^k$ and only work on coefficients in the support set. Moreover, we do not need to compute $\Omega(\text{SPGS}(z))$ ex-
actly. For separable, sparsity-inducing $g$, we can simply predict the support set by keeping coefficients with the larger gradients. Specifically, we can estimate the support set by

$$
\hat{\Omega}_z = \{ i \mid i \in \Omega(z) \text{ or } a_i > \frac{1}{2}(\|a_{\Omega(z)}\|_\infty + \|a\|_\infty) \}
$$

(4.11)

where $a = (\text{sign}(z) \cdot \partial f(z))^+$ can be interpreted as the force that pulls the coefficients away from 0.

![Figure 4.1: An illustration of gradient pruning, in which red and blue arrows denote $\partial f$ and prune($\partial f, 1$), respectively. For $z^0$, the pruned gradient keeps the largest component. For $z^1$, the largest gradient component is still in the same direction. For $z^2$, the largest gradient component is outside of the support set, so both components are kept after pruning. With a full support set, there is no further delta pruning at $z^3$.](image)

Unlike the computation for $\tau$ in (4.6), the predicted support set in (4.11) only depends on the gradients and not the output of SPGS. This leads to a simplified version of SPGS where the only modification from PGS is the pruning of gradients, as shown in Algorithm 9. The pruning function removes any gradient components that
are not in the estimated support set:

\[ \text{prune}_z(g) = \begin{cases} g_i & i \in \hat{\Omega}_z \\ 0 & i \notin \hat{\Omega}_z \end{cases} \quad (4.12) \]

The **SPGS** for separable regularization can be viewed as block coordinate descent, because only the coordinates in \( \hat{\Omega}_z \) are updated. Specifically, we can rewrite the \( y \) update in Algorithm 9 as

\[ y_{\hat{\Omega}_z} := \text{prox}_{\frac{1}{L} g}(x_{\hat{\Omega}_z} - \frac{1}{L} \partial f(x)_{\hat{\Omega}_z}) \quad (4.13) \]

For separable regularization, it is easy to see that ASH enhanced PG retains the linear convergence rate.

---

**Algorithm 9** **SPGS** for Separable Regularization

*input* \( x, L, \tau, \) parameter \( \eta \)

*repeat*

- Compute \( \hat{\Omega}_z \) using (4.11).
- \( y := \text{prox}_{\frac{1}{L} g}(x - \frac{1}{L} \text{prune}_\tau(\partial f(x))) \)
- \( L := \eta L \)

*until* \( Q(y) \leq Q(x) \)

*output* \( y, \frac{1}{\eta} L \)

---

**Lemma 2** (Convergence). *Let \( z^0 = 0 \) and define \( z^k = \text{SPGS}(z^{k-1}) \). Then*

\[ Q(z^k) - Q(z^*) \leq \frac{2nLR^2}{k + 4} \]

*where \( R \geq \|z^0 - z^*\| \).*

45
Proof. We can use the proof derived by Nesterov [52].

\[
Q(z^k) - Q(z^{k+1}) \geq \frac{1}{2L} \|\partial f(z^k)\|^2_{\Omega(z^k)}^2 \geq \frac{1}{2nL} \|\partial f(z^k)\|^2 \\
\geq \frac{1}{2nL} (Q(z^k) - Q(z^*))^2
\]

\[\Box\]

4.4.2 ASH FOR COORDINATE DESCENT

In cyclic and random coordinate descent (CD), the coordinates are updated sequentially. We can apply the same idea of following a sparse path by only updating coefficients within a subset. Since CD only works for separable regularization, we can use (4.11) to compute the estimated support set. The ASH modified CD is shown in Algorithm 10. Similar to standard coordinate descent, one can also update \( z_i \) using closed-form solutions if it can be efficiently computed.

**Algorithm 10 ASH Coordinate Descent**

\[\text{Initialize } z = 0 \in \mathbb{R}^n\]
\[\text{repeat}\]
\[\text{update } \tau\]
\[\text{for each } i \in \hat{\Omega}_z \text{ do}\]
\[z_i := \text{prox}_{\frac{1}{L} g} \left( z_i - \frac{1}{L} \partial f(z)_i \right)\]
\[\text{end for}\]
\[\text{until } \text{termination condition is met}\]
\[\text{output } z\]
4.5 Characteristics of ASH

Our strategy is to build up support sets more carefully by pruning updates. While being algorithmically similar to gradient descent methods, ASH tends to work on important coefficients early on and terminate without ever updating the majority of coefficients. We illustrate the difference between LARS, FISTA, FSS, and ASH:FISTA (FISTA modified with ASH) by showing the intermediate solutions $z^k$ in solving a small scale LASSO in Figure 4.2.

![Graph showing intermediate solutions $z^k$ for different solvers.](image)

**Figure 4.2:** The intermediate $z^k$ of different solvers in the first 10 iterations in solving a small LASSO problem. FISTA starts off with very dense estimates and gradually converges to a sparse solution. ASH is nearly identical to FISTA in terms of algorithm and computational complexity, yet it maintains an exactly sparse support set similar to LARS. FSS has a similar pattern since it also gradually accumulates the support set. Note that LARS and FISTA reach the larger coefficients gradually, whereas FSS and ASH tend to overshoot them in initial iterations. In this example $A \in \mathbb{R}^{150 \times 200}$ and $\|z^*\|_0 = 10$.

The most notable characteristic of ASH is that there are only very few active coefficients throughout the iterations, whereas the unmodified FISTA has very dense interim solution. The ASH solution path is more similar to those from FSS or LARS. However, unlike LARS and FSS, ASH sometimes adds multiple coefficients into the support set in a single iteration. In this example such expansion of support set allows ASH to converge faster because those coefficients all turned out to be in the support
set of $z^*$.

![Graph](image)

**Figure 4.3: Support Set Expansion.** LASSO problem with $A \in \mathbb{R}^{700 \times 1000}$ with 3 different levels of sparsity. The speed of expansion depends on the sparsity of final solution. The colored dotted and solid lines represents FISTA and ASH:FISTA, respectively. The black straight line indicates the expansion bound of FSS and LARS (they can only add increase support set by 1 per iteration).

Figure 4.3 further illustrates the change in support set over iterations. Note that ASH expands the support set adaptively. FISTA tends to include many coefficients in the beginning, then gradually reduces the support set through the proximal or shrinkage operator. In some sense, FISTA tries to deal with too many coefficients at the same time, and FSS is too conservative in the way they expand the support set.

### 4.6 Summary and Remarks

The auxiliary sparse homotopy (ASH) method modifies the existing gradient descent methods so that they follow a sparse path towards the solution. ASH can be interpreted as homotopy a method where we insert an auxiliary $\ell_0$ constraint as a stepping stone. Alternatively, it can be interpreted as a working set method with a special support set expansion rule. The homotopy view shows that ASH is applicable to general sparse problems, and the working set view provides intuition on how the sup-
port set should be expanded (which corresponds to setting $\tau$, the auxiliary homotopy regularization parameter).

We claim the following two contributions: (1) A general path-following strategy by introducing additional $\ell_0$ constraint as stepping stone, and (2) A support set expansion strategy that is parameter free and adaptive.
In this section, we consider the optimization problems outlined in Section 1.5 and evaluate the performance of ASH against other solvers. The speed is measured by iterations instead of actual running time in order to have the reported numbers be independent to the actual implementation. We consider this fair because ASH iterations are at least as fast as all the alternative methods.

We also provide empirical results to demonstrate that sparse modeling is indeed effective in image classification, compressive sensing, and feature selection.
5.1 The LASSO Problem

In this section we compare the performance of different methods at solving the LASSO. FISTA [4]\(^1\) and *proximal quasi-Newton* (PQN) [6] represents proximal methods using full gradient. The *Feature sign search* algorithm (FSS) [38] and *matching pursuit* LASSO (MPL) [66] represent working set methods. The *proximal gradient homotopy* method (PGH) [75] represents homotopy methods. Coordinate descent (CD) [59] is represented by randomized CD. Other methods such as FPC-AS, IFB, mAP, reweighted \(\ell_2\), GPSR and spaRSA are not included in the comparison because at least one of the above methods is on par with them. Finally, we use ASH to modify FISTA and CD as explained in Section 4.

![Figure 5.1: Medium sized LASSO problem with \(A \in \mathbb{R}^{2000 \times 20000}\) with \(\|\mathbf{z}^*\|_0 = 200\). CD, ASH:CD and ASH:FISTA converges at around the same number of iterations. Note that CD and FISTA iterations use much larger support sets, so they have higher cost per iteration. MPL is sensitive to the initialization of constant \(\rho\), which result in a large variance in its performance.](image)

Figure 5.1 shows the result for a medium scale LASSO with a design matrix of size \(2000 \times 20000\) and sparsity of the solution being around 1\% (\(\sim 200\) nonzeros). ASH delivers the best performance. For unmodified FISTA and CD, the support set starts

\(^1\)Note that we use the monotonic FISTA in (2.18), which is similar to FISTA with adaptive restart proposed by Candes et al. [55].
out dense and gets slowly refined into the correct support set. For all other methods (working set, homotopy, and ASH), the support set grows until it covers the desired set. MPL is sensitive to the initialization of constant \( \rho^2 \), which result in a large variance in its performance.

Figure 5.2: Convergence comparison for solving LASSO. ASH delivers the best performance overall. FISTA and CD do better when correlation in the design matrix \( \Lambda \) is low (tall matrices). Working set methods (FSS and MPL) does well when the minimizer is very sparse. Note that ASH consistently outperforms other methods that explicitly exploit sparsity of the solution, namely FSS, MPL and PGH.

Figure 5.2 presents convergence rate for a small scale LASSO under four settings:

(a) wide matrix, moderate \( \ell_0 \)-norm, (b) tall matrix, moderate \( \ell_0 \)-norm, (c) wide matrix, small \( \ell_0 \)-norm, (d) tall matrix, small \( \ell_0 \)-norm. Tall matrices are sized \( 2000 \times 500 \) and wide matrices \( 500 \times 2000 \). ASH is consistently better than its unmodified counter-

\( ^2 \)The constant \( \rho \) in MPL is determined by the gradient at the initial point \( z^0 = 0 \). See Eqn (24) in the MPL paper [66].

52
Figure 5.3: Number of iterations required to reach a target precision for different $\lambda$. FISTA and CD converges faster with stronger regularization as discussed in Section 3.3. The working set and homotopy methods are insensitive to the choice of $\lambda$. For moderate precision, ASH is around $2 \times$ faster than other methods.

parts for wide matrices. For tall matrices, all methods converge very fast because the problem becomes easier, except for FSS since it needs at least $k$ iterations to build up the support set.

Figure 5.3 illustrates the impact of $\lambda$ in convergence speed under setting (a)$^3$. FISTA and CD becomes slower when $\lambda$ is smaller. On the other hand, the performance of working set and homotopy methods are quite insensitive to the choice of $\lambda$ (they do depend on the sparsity of solution, which is indirectly related to $\lambda$).

In most applications, we do not need to solve LASSO with high precision. In Figure 5.3, we consider the number of iterations required for each method to reach a target precision, and show that ASH:FISTA is at least $2 \times$ faster than other methods. Upon reaching moderate precision, the improvements in objective function value is already visually indistinguishable if we use Figure 5.2 for reference$^4$.

To summarize: ASH is faster than the following methods in our experiments: CD [59, 73], PG Homotopy [75], MPL [75], FISTA [4, 55], Nestrov’s APG [53], mAP [39], and

$^3 A \in \mathbb{R}^{500 \times 2000}, \|z^*\|_0 = 50$

$^4$Distance to $Q(z^*)$ is within $10^{-4}$ and $10^{-8}$ for moderate and high precision, respectively.
GIST [32]. It has already been shown in the literature that FPC-AS [70], GPSR [29], spaRSA [74], and IFB [11] are outperformed by at least one of the methods listed above; therefore we did not directly compare against them.

5.2 Sparse Coding in Hierarchical Models

Hierarchical models based on sparse coding have been shown to do well on some classification tasks [21, 76, 78]. Each layer in the model transforms the data into some kind of higher-level representation using sparse coding. Specifically, an input $x^t$ at the $t$-th layer would be transformed by $x^{t+1} = \text{SparseCode}_{D'}(x^t)$ where $D'$ is the dictionary associated with this layer. A supervised classifier sits on top of the model and takes in the high-level representation as input to make the final predictions. This model is mostly unsupervised, because the weights/dictionaries in the lower layer can be trained without labels.

There are several components in these types of models: patching, dictionary learning, sparse coding, representation aggregation, and the classifier. Next, we go through each of these components.

Patching

An input will often be broken down into patches and encoded independently. The size of patches and sliding window step size often has a large impact on the prediction accuracy.

Dictionary Learning

A dictionary is trained to model the inputs parsimoniously as a sum of few features. Let $\{x_0, x_1, \ldots, x_N\}$ with $x_i \in \mathbb{R}^p$ be a set of training patches. We learn a dictionary
by solving:

$$D = \arg \min_{D \in \mathbb{R}^{p \times n}, z_0, z_1, \ldots, z_N} \sum_{i} \left\{ \|x_i - Dz_i\|_2^2 + g(z_i) \right\}$$

where $N$ is the number of training patches, and $g$ is a sparsity-inducing regularization. A column in $D$ is called an atom. Dictionary learning works due to the fact that in order to minimize the reconstruction error, each atom needs to represent some recurring pattern. The dictionary $D \in \mathbb{R}^{p \times n}$ tends to be overcomplete ($p < n$) in order to adequately capture variations in data.

Solving (5.1) directly is difficult, and it is more efficient to break it into alternating subproblems. The most straightforward approach is to solve for $\{z_i\}$ while keeping $D$ fixed (sparse coding), and then solve for $D$ while keeping $\{z_i\}$ fixed (dictionary update). It has been shown that updating the dictionary after computing the sparse code for a few samples can significantly improve the dictionary learning cost [43, 44]. The performance of the dictionary learning largely depends on the speed of sparse coding, since the dictionary update step tends to be relatively cheap.

The choice of $g$ greatly impacts the final performance. In most cases, using the $\ell_1$ penalty yields better results than the $\ell_0$ constraint. Another choice of $g$ is to use structured sparsity or group sparsity as described in Section 1.2. In particular, it has been shown that interesting structures can be learned by using overlapping group sparsity when training a dictionary, as shown in Figure 5.4.
Figure 5.4: Dictionary trained with $\ell_1$ constraint (left) and overlapping group sparsity (right). Group sparsity encourages neighboring atoms to be selected together, therefore the atoms that are close-by tend to represent similar patterns. See [46] for more details.

**Sparse Coding**

Sparse coding transforms its input and represents it into a sum of features. Given a dictionary $D$, the sparse coding is defined as:

$$\text{SparseCode}_D(x) \equiv \arg\min_z \|x - Dz\|_2^2 + g(z) \quad (5.2)$$

where $g$ is a sparsity-inducing regularization. This regularization does not necessarily have to match with the one used for dictionary learning.

Sparse coding with $\ell_1$ is the LASSO problem. We have shown in Section 5.1 that ASH outperforms other methods in solving LASSO under the sparse coding setting (where design matrix is wide). Another choice of $g$ is structured sparsity. In Figure 5.5 we show that ASH again improves FISTA and PGH, which is the state-of-the-art for this type of regularization [1]. MPL does not perform well because the expansion rule assumes separable regularization, and does not take the sparse structure
into account. The proximal operator is evaluated using ProxFlow [45].

![Graphs of Objective and Support Set](image)

**Figure 5.5:** Overlapping group LASSO with \( D \in \mathbb{R}^{500 \times 2000} \) atoms. The group norm is defined as \( g(z) = \sum_{g \in G} \| z_g \|_{\infty} \), with contiguous pattern (See Figure 1.3). MPL doe outperforms.

### Aggregation

Each input sample is initially broken down into many patches and coded independently, so we need to aggregate the pieces before reaching the classifier. A common aggregation method is max pooling [60], which takes \( p \) input patches \( \{ x^0, x^1, ..., x^p \} \) and finds the largest coefficient among the patches in each dimension as its output:

\[
[x]_i = \max \{ [x^0]_i, [x^1]_i, ..., [x^p]_i \}
\]

Similarly, one can replace \( \max \) with other functions such as average pooling [12].

### Classifier

After layers of feature transformation, the data should be represented in an abstract format that makes prediction easier. It is therefore common to use simple classifiers such as linear support vector machine (SVM) or logistic regressor (LR).
5.2.1 Example: Emotion Classification

In this section, we consider the emotion classification problem where we are interested in deciding whether a subject in a photo is smiling or not. As shown in Figure 5.6, these photos can have large variations in posture and lighting conditions.

![AM-FED](image1) ![GENKI](image2) ![CK+](image3)

**Figure 5.6:** Examples from three smile classification datasets: AM-FED [47], GENKI [72] and CK+[41]. AM-FED and GENKI contains wild images with large variations in lighting and posture. The photos in CK+ are all staged.

We use a model with single hidden layer, similar to the one presented by Coates et al. [21]. We fist split an input image into the left and right patch. Then, we apply a novel mirroring technique so that both patches have the same orientation when they are being encoded. After encoding the two patches, we use max pooling to aggregate the sparse codes before sending it to the final linear SVM classifier. Figure 5.7 shows this pipeline of transforming the input into a feature vector.

We compare several encoder variations in our method. First, we consider the $\ell_1$ penalty and the $\ell_0$ constraint for sparse coding. We use OMP to solve the sparse coding problem in the $\ell_0$ case\(^5\). Secondly, we consider adding the non-negativity constraint on the sparse codes. It has been shown that non-negativity constraint im-

\(^5\)Under $\ell_0$ penalty, the problem is non-convex and different solvers may find different sparse codes. We choose OMP because they have been successfully used in the literature [40].
proves the stability of sparse codes for classification purposes [40]. Lastly, we compare the performance with and without the mirroring technique.

Figure 5.7: A simple sparse coding based classification model.

Figure 5.8: Smile classification task using different sparse coding settings. The performance of the models scales with the amount of training data. The dataset used in this experiment is GENKI [72].

Table 5.1 shows the performance of these models. Note that our method using sparse modeling outperforms the prior best results that are based on dense representations. LASSO outperforms OMP when other things are held equal, and the non-negativity constraint and mirroring both improve the prediction accuracy. It is worth noting that OMP with non-negativity constraint and mirroring is significantly better than raw OMP, and is almost competitive against LASSO with all the modifications.

59
Table 5.1: Performance results in area under the ROC curve (A’) comparing LASSO and OMP sparse coding methods on datasets containing “happiness” and “neutral”. Note that MNNSC-LASSO performs the best on both datasets, with MNNSC-OMP performing comparably.

<table>
<thead>
<tr>
<th>Method</th>
<th>GENKI-4K</th>
<th>AM-FED</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNNSC-LASSO</td>
<td>97.0</td>
<td>92.3</td>
</tr>
<tr>
<td>NNSC-LASSO</td>
<td>96.7</td>
<td>91.2</td>
</tr>
<tr>
<td>SC-LASSO</td>
<td>95.1</td>
<td>89.7</td>
</tr>
<tr>
<td>MNNSC-OMP</td>
<td>96.2</td>
<td>92.1</td>
</tr>
<tr>
<td>NNSC-OMP</td>
<td>95.7</td>
<td>88.8</td>
</tr>
<tr>
<td>SC-OMP</td>
<td>93.1</td>
<td>86.0</td>
</tr>
<tr>
<td>Gabor</td>
<td>95.7</td>
<td>88.9</td>
</tr>
<tr>
<td>Best Reported</td>
<td>96.1 [71]</td>
<td>90.0   [47]</td>
</tr>
</tbody>
</table>

Figure 5.8 shows how each setting scales with the amount of training data.

5.3 Compressive Sensing

Compressive sensing is a way to sample signals at rates proportional to actual information by exploiting its sparsity.

Let \( x \in \mathbb{R}^n \) be a sparse signal such that \( x = Dz + \epsilon \) for some basis \( D \in \mathbb{R}^{n \times t} \) and \( z \in \mathbb{R}^t \) with \( \|z\|_0 \leq k \) for some \( k \ll t \). We assume the residual \( \epsilon \) is small. The measurements \( y \in \mathbb{R}^m \) of a signal \( x \) are obtained through linear projection \( \Phi \):

\[
y = \Phi x
\]

(5.4)

where \( \Phi \in \mathbb{R}^{m \times n} \) is called the sensing matrix\(^6\). Based on the assumption that \( x \) has a sparse representation with respect to \( D \), the original signal is recovered by solving the following optimization problem:

\[
\hat{z} = \arg \min_z \|y - \Phi Dz\|_2^2 + \lambda g(z)
\]

(5.5)

\(^6\)The sensing matrix if also called the sampling or measurement matrix.
where \( g \) is the \( \ell_1 \) or \( \ell_0 \) penalty/constraint. Then, we can recover \( x \) with \( \hat{x} = D\hat{z} \).

Assume that any column subset of \((\Phi D)\) with size less than \(2k\) approximately behaves like an orthogonal system (this is known as the restricted isometry property RIP). Then the signal can be recovered with high probability for \( z \) satisfying \( \|z\|_0 \leq k \) \([15, 16, 23]\). It has been shown that for a randomized sensing matrix \( \Phi \) that satisfy RIP, we can select an orthonormal basis \( D \) such as the Fourier or wavelet basis, and \((\Phi D)\) would still satisfy RIP with high probability. Empirically, we find that signals can be recovered in a more relaxed setting that uses an overcomplete dictionary \( D \) instead of an orthonormal basis.

This scheme is called compressive sensing due to the fact that the number of measurements \( m \) required for robust signal recovery mostly depends on \( k \), making it much smaller than \( n \) (or \( t \)). Therefore, the measurements \( y \) are a compressed version of \( x \). Compressive sensing is most effective in applications where the sensing matrix can be implemented efficiently. Some notable examples that physically implement \( \Phi \) includes the single pixel camera \([24]\) and compressive MRI \([42]\).

### 5.3.1 Solving the Compressive Sensing Problem

With the \( \ell_1 \) regularization, the problem in (5.5) becomes the LASSO \([67]\). There are many efficient solvers for the LASSO as mentioned earlier in Section 5.1. Candes and Tao \([16]\) and Donoho \([23]\) first derived the signal recovery guarantees and showed that under RIP assumptions, minimizing \( \ell_1 \) is the same as minimizing \( \ell_0 \).

The optimization problem in (5.5) is combinatorial with the \( \ell_0 \) constraint, but we can use greedy algorithms such as orthogonal matching pursuit (OMP) \([68]\), CoSamp \([48]\), and iterative hard thresholding (IHT) \([10]\), etc. Some methods have stronger guarantees on signal recovery rate, but in practice the performance of these solvers are often
much better than the lower bounds given by theoretical analysis.

We can modify IHT with ASH. The IHT algorithm is basically a special case of proximal gradient descent algorithm, where the proximal operator is the hard-thresholding function. By applying ASH to IHT, we also observe faster convergence speed that is similar to the results in Section 5.1.

The compressive sensing problem with $\ell_0$ constraint is non-convex, so different algorithms can converge to different solutions. It turns out ASH:IHT has a better chance of recovering the correct signal when compared to baseline IHT. Figure 5.9 shows the histogram of incorrectly recovered support using OMP, IHT and ASH:IHT.

![Figure 5.9: Histogram of incorrectly recovered support. OMP tends to get most of the support set right, and has low variance in its performance. On the other hand, IHT tends to have two modes — it either gets the entire support set right, or it gets many of them wrong. ASH:IHT outperforms IHT.](image)

### 5.3.2 Nonlinear compressive sensing

In the compressive sensing framework, the measurements $y$ is modeled as a linear projection of the signal of interest. We can extend this framework to handle distorted measurements in physical devices. For example, power amplifiers are only linear for a specific input/output range. Unfortunately, that range also happens to have lower
power efficiency (See Figure 5.10). If one can tolerate distortion from a nonlinear power amplifier, then the amplifier may be able to operate with a much higher efficiency.

![Graphs showing amplifier nonlinearity at different output power levels](image)

(a) 24.5 dBm, 9.25% Eff.  (b) 28.1 dBm, 16.3% Eff.  (c) 31 dBm, 24.5% Eff.

**Figure 5.10:** Amplifier nonlinearity in three different output power levels (and power efficiency). Power amplifiers are normally designed to have a linear region so that the output is has a linear gain with respect to the input. However, they are more efficient when operated in the nonlinear region of the characteristic curve (higher output dBm with saturation). The outputs are rescaled by the amplifier gain. A 3rd degree polynomial is used to model the nonlinearity.

![Graphs comparing simulated and real measurements](image)

(a) Simulated Distortion  (b) Real Measurements

**Figure 5.11:** Reconstructing signal from distorted measurements using the standard compressive sensing formulation and the nonlinear extension. Standard compressive sensing treats the nonlinearity as noise, which results in a lower SNR. (a) Simulated distortion using \( T(x) = \frac{1}{\alpha} \tanh(\alpha x) \). (b) Measured distortion by operating a power amplifier in its nonlinear region.

We consider the following measurement model for nonlinear compressive sensing:

\[
y = T(\Phi x)
\]  \hspace{1cm} (5.6)
where $T(\cdot) : \mathbb{R}^m \rightarrow \mathbb{R}^m$ is the nonlinear distortion function. For example, we can roughly model the saturated outputs from amplifiers using a $\tanh$ function or a 3rd degree polynomial, as shown in Figure 5.10.

![Graphs showing convergence of different methods](image1.png)

**Figure 5.12:** The ASH:FISTA and ASH:mAP methods both converge faster than their standard counterparts for linear and non-linear compressive-sensing reconstruction.

In this context, it is natural to incorporate the distortion model when we try to recover the signal. We call this the nonlinear compressive sensing (NLCS):

$$
\hat{z} = \arg\min_z \|y - T(\Phi Dz)\|_2^2 + \lambda g(z)
$$

(5.7)

Note that unlike standard CS with $\ell_1$ penalty/constraint, the NLCS problem is non-convex due to the distortion function.

Figure 5.11 shows a comparison of the reconstruction SNR using CS and NLCS on simulated and measured distortion. NLCS delivers significantly better SNR in both cases. We use the $\ell_1$ constraint formulation. In terms of convergence speed, Figure 5.12 shows that ASH modified mAP and FISTA both converge faster than their unmodified counterparts.
### 5.4 Sparse Logistic Regression

In this section, we use a real-world medical dataset to evaluate the training of a logistic regressor under different penalization functions. The classification task associated with this dataset is to determine whether a patient has Acute Myeloid Leukemia (AML) or Acute Lymphoblastic Leukemia (ALL). We train the sparse logistic regression classifiers on 72 subjects with 7128 gene-expression each. For this application, sparsity encodes the expectation that only a small number of genes are related to AML and ALL.

Our sparse logistic regression formula is given by:

$$
z = \arg \min_z \frac{1}{N} \sum_{i}^{N} \log(1 + \exp(-y_i x_i^T z)) + g(z) \tag{5.8}$$

were \((x_i, y_i) \in (\mathbb{R}^n, \mathbb{R})\) are the training samples. We consider four types of penalty functions: (1) capped \(\ell_1\), (2) log sum, (3) Smoothly Clipped Absolute Deviation (SCAD), and (4) minimax concave penalty (MCP).

The general iterative shrinkage and thresholding methods (GIST) \(^{[32]}\) is one of the fastest non-convex optimization toolsets for solving these problems. In this experiment, we use the GIST software\(^7\) and extend it with ASH. Figure 5.13 shows that our method converges with fewer iterations than GIST, and uses smaller support sets throughout iterations. Note that in all cases the ASH version finds a solution with lower objective and fewer nonzeros.

To our best knowledge, sparse logistic regression produces the state-of-the-art classification results for this dataset in terms of accuracy. As shown in Table 5.2, sparse

\(^7\)http://www.public.asu.edu/~jye02/Software/GIST/
Figure 5.13: Comparison between GIST and ASH-GIST on sparse logistic regression with different types of regularization functions. First row: objective in log scale. Second row: size of support set. The parameters are chosen so that the final solution only has 10 to 20 nonzero coefficients. The prediction accuracy of the capped $\ell_1$ model is shown in Table 5.2.

Table 5.2: Classification between acute myeloid leukaemia (AML) and acute lymphoblastic leukaemia (ALL) based on gene expression. The data contains expression of 7128 genes. Sparsity corresponds to the number of genes that are selected by the model for prediction. Lower sparsity means the predictions are made with a more concise model, thereby reducing the risk of over-fitting. Cawley and Talbot evaluated their model using leave-one-out training instead of the default train/test partitions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Sparsity</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLogReg [19]</td>
<td>11.59$^*$</td>
<td>93.05</td>
</tr>
<tr>
<td>BN [77]</td>
<td>18</td>
<td>85.29</td>
</tr>
<tr>
<td>BN-NC [77]</td>
<td>24</td>
<td>97.05</td>
</tr>
<tr>
<td>ASH: sparse SVM</td>
<td>15</td>
<td>97.05</td>
</tr>
<tr>
<td>ASH: sparse LR</td>
<td>12</td>
<td>97.05</td>
</tr>
</tbody>
</table>

LR with MCP uses only 12 genes to make accurate predictions when solved with ASH (when trained with GIST, it uses 18 genes). The previous best record uses 24 genes to achieve the same accuracy [77]. This means ASH not only finds a solution faster, but the quality of solution is also better when smaller $\ell_0$ norm is desired.
5.5 Sparse Neural Nets

Training a neural net can also be viewed as an optimization problem, but with many latent variables. The fitness function for a neural net can be described by:

$$f(w) = \sum_i \text{loss}(y_i, \text{feed-forward}(w, x_i))$$  \hspace{1cm} (5.9)

where the parameter $w$ represents the weights of connections in the neural net. Without going into any details, the key point is that computing the gradients of $f$ with respect to $w$ is easy. Neural net training is done through back-propagation, which is essentially using stochastic gradient descent to minimize $f$.

Standard neural nets are very inefficient in terms of neural connections, because they are very dense. It has been shown that a network can often do almost equally well with more than 90% of the connections removed [34, 35]. If the cost of a neural net depends on the number of connections and not just the number of neurons, then it may be desirable to train sparse neural networks by regularizing the connection weights. There have only been very few attempts in addressing this issue, which is likely due the fact that existing computing models cannot exploit the sparse connections effectively. However, considering that real neural nets (brains) are all sparsely connected [61], it seems like sparse neural nets would be the right direction to go.

In this experiment we consider neural nets for classifying the MNIST handwritten digit dataset. In particular, we are interested in adding $\ell_0$ constraints to the weights, so that the learned connections would be sparse. We use a fully connected model with two hidden layers, and constrain the network so that only 10% of the weights are active in each layer. Figure 5.14 shows an example of a sparsely connected network.
We tried many different solvers for training the network, including momentum SGD [62], RMSprop [22, 33], Nesterov’s accelerated gradient method [8], adaGrad [25], adaDelta [79] and ADAM [37]. Since all of them are gradient methods, we can incorporate the $\ell_0$ constraint by simply adding a hard-threshold function after the updates (which turns these gradient descent methods into proximal gradient descent). There were no significant differences in the prediction accuracy of networks trained with different methods. ADAM has the fastest convergence speed overall, so we will focus on it in the remainder of this section.

We compare ADAM and its ASH modified version. As expected, ASH:ADAM outperforms ADAM in terms of convergence speed. Interestingly, ASH also seems to find better solutions that give lower objective value and better prediction accuracy. We trained 20 networks with different numbers of neurons using ADAM and
ASH:ADAM. Figure 5.15 shows that ASH consistently outperforms ADAM in every setting.

Figure 5.16 shows the prediction error and the number of active connections in different models. The number of neurons in each layer are selected at random, so there is a high variance in the performance. However, there is a clear trend that larger networks tend to perform better, and that sparse networks performs better than dense networks.

5.6 Summary and Remarks

We compared the convergence rates of the most up-to-date methods in solving LASSO. Working set methods typically perform better when the solution is very sparse, because in those cases the support set can be identified rather quickly. MPL has high performance variance due their sensitivity to its tuning parameter and its initializa-
Figure 5.16: The performance of dense network, sparse network, and sparse networks trained with ASH. Larger network almost always delivers better performance, but sparse networks can be more efficient than dense networks because it can express a richer model (assuming no over-fitting).

In addition, we present several applications that involves solving sparse optimization problems. In all of these applications, ASH outperforms the state-of-the-art solvers for each optimization problem. This includes FISTA [45] for structured sparse coding, IHT [10] for $\ell_0$ nonlinear compressive sensing, mAP [39] for box-constraint com-
Discussion

In this section we try to summarize the characteristics of three majors types of solvers, and address some computational concerns.

6.1 Brief Summary of Different Methods

1. Proximal gradient methods: ISTA, FISTA, mAP

   Proximal methods are very general and work with any convex, differentiable $f$ and convex $g$. When $f$ is strongly convex, accelerated proximal methods speedup automatically. In the non-convex case, these proximal methods can still converge to a local optimal. These methods do not utilize the fact that the
solution is sparse.

2. **Homotopy and active set methods: LARS, FSS, PHG, MPL**

Homotopy and working set methods typically converge faster than proximal methods for small problems or when the solution is very sparse. The cost per iteration tends to be low because they work on small support sets. Their performance depends largely on the support set expansion strategy and innerloop solvers. For example, in LARS and FSS the innerloop is a small least squares problem that can be solved efficiently. However, they both expand support set very slowly.

Existing working set methods may not work well for problems with non-separable regularization. Typical expansion rules select the component with largest immediate gain (e.g., largest gradient in the case of FSS and OMP), but this principle only makes sense when the regularization cost of a coefficient is independent to the state of other coefficients.

As explained in Section 3, existing Homotopy methods tend to be too conservative due to bias from large regularization. Also, they are sensitive to parameters controlling the regularization relax rate and innerloop precision.

As described in Section 4.3, ASH can be viewed either as a homotopy method or a working set method. However, ASH can handle non-separable regularization, while avoiding the bias in regular homotopy methods by introducing the auxiliary constraint.

3. **Coordinate descent**

Despite their simplicity, coordinate descent methods are competitive in solving sparse problems. However, we note that random or cyclic CD may waste
effort in updating coefficients that are eventually irrelevant, but they are still faster than full gradient methods in solving sparse problems. The greedy version selects the coefficient more carefully, but it sacrifices parallelizability and therefore does not scale. The main drawback of CD is that it can only deal with separable $g$ (e.g., LASSO and sparse logistic regression).

ASH for separable regularization as presented in Section 4.4.1 can be viewed as a greedy block coordinate descent (BCD) method with dynamic block selection. As presented in Section 4.4.2, ASH can also be applied to cyclic CD and randomized CD to reduce the number of coefficient updates. In both cases, ASH only pays the selection overhead per $|\hat{\Omega}_z|$ updates\footnote{Recall that $\hat{\Omega}_z$ is the predicted support set described in (4.11).}, as opposed to once per coefficient update in the case of greedy CD.

6.2 Computational Gains from Sparsity

Computation with sparse vectors is much cheaper than dense vectors in terms of both space and time. Encouraging sparsity updates can therefore create a big computational advantage.

For first order gradient methods, the cost per iteration is dominated by the gradient update (assuming the proximal operator is cheap). Consider the gradient computational cost in solving LASSO in (1.15). For design matrix $A \in \mathbb{R}^{p \times n}$, the gradient of squared loss $\|x - Az^k\|_2^2$ at interim state $z^k$ is $\partial f(z^k) = 2(A^T x - (A^T A) z^k)$. Assuming $A^T x$ is already computed, this means the real effort is computing $A^T A z^k$.

In some cases we can precompute the Gram matrix $A^T A$ to save some computation for sparse $z^k$. As shown in Table 6.1, the cost for computing the gradient for a
Table 6.1: Cost for computing the gradient of the squared-loss function.

<table>
<thead>
<tr>
<th>task</th>
<th>cost (dense)</th>
<th>cost (sparse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ (A^T A)<em>{\Omega z</em>{\Omega}}$</td>
<td>$n^2$</td>
<td>$nK$</td>
</tr>
<tr>
<td>$ A^T (A_{\Omega z_{\Omega}}) $</td>
<td>$2pn$</td>
<td>$pn + pK$</td>
</tr>
</tbody>
</table>

$k$-sparse update is $\frac{n}{k}$ times cheaper than that of a dense one. This can easily be several orders of magnitude when the solutions are very sparse\(^2\).

6.3 Parallelizability

Parallelization is a key for solvers to handle larger problems. Most gradient descent methods are scalable, because both the gradient computation and the step update can be computed in parallel. ASH methods are equally as easy to parallelize.

Consider the LASSO problem in (1.15). Assume we have multiple computation nodes, and that $z^k$ is already distributed to each node. Assume that the rows of $A^T A$ is evenly distributed among the nodes, and each node is responsible for updating the coefficients corresponding to the rows in $A^T A$.

Computing the proximal gradient step is embarrassingly parallel: the nodes can independently compute $[z^{k+1}]_s = \text{prox}_g([z^k]_s + 2A^T_s(x - Az^k))$ where $s$ is the set of coefficients that the node is responsible for. After these updates, the nodes would need to broadcast $[z^{k+1}]_s$ or $([z^{k+1}]_s - [z^k]_s)$ to all other nodes for the next iteration. This communication cost is proportional to the number of nonzero updates, so it can be reduced if the updates are sparse.

The parallelizability of active set methods depends almost entirely on the innerloop solver. One can always use gradient methods to solve the innerloop when parallelization is important.

\(^2\)See the leukemia classification example in Section 5.4.

75
Coordinate descent is harder to parallelize, since the updates are sequential. There are many recent developments on this topic [13, 28, 59]. In short, CD can be parallelized by making them more similar to the full gradient methods: the parallelization basically comes from updating more coefficients at the same time\(^3\).

6.4 Benefit of \(\ell_0\)-Sparse Solutions in Non-convex Problems

When the optimization has multiple local optima, working set methods tends to converge to those that are sparser in the \(\ell_0\) sense. This is hardly surprising, considering that these methods have interim solutions that is \(\ell_0\)-sparse.

The tendency to find exactly sparse solutions could be very desirable in the context of feature selection, in which we ideally want to find few predictive factors with respect to some tasks. For example, in the cancer classification problem the nonzero coefficients correspond to genes that are identified for further study [19]. Another example is the L1000 gene expression data project, in which 1000 out of the 20000 human genes were found to capture roughly 80% of the information [58]. In addition, models with exactly sparse parameters are easier to interpret, which is very important for tasks that involve human in the loop [17].

6.5 Summary and Remarks

Methods that have sparse interim solutions have a computational advantage, because operations associated with the zero coefficients (e.g., when updating the gradient) can be dropped. ASH, PGH, FSS, CD and MPL enjoy computational gains from sparsity.

The gradient methods are easy to parallelize, with the exception of CD due to its

\(^3\)One can also use asynchronous updates, but this is beyond the scope of this thesis.
sequential nature. Some working set methods may have innerloops that are difficult to parallelize, but it is usually possible to swap out the innerloop solver for something that scales better. ASH, FISTA, PGH, MPL are easy to parallelize.
Optimization problems with sparsity-inducing regularization are at the core of many modern algorithms. There has already been a substantial amount of work since sparse models became popular in the last two decades. However, solving the general sparse optimization problem efficiently remains to be an interesting challenge.

In this thesis, we propose a strategy to improve gradient descent methods when solving general sparse optimization problems. We present the *auxiliary sparse homotopy* (ASH) method that modifies existing gradient descent methods by encouraging them to traverse a sparse solution path. The main designs are:

- A general path-following strategy applicable to sparse optimization problems.
• A support set expansion strategy that is parameter free and adaptive.

ASH has fast convergence by explicitly exploiting the fact that solutions are sparse, and low computation cost per iteration due to small interim support sets. Moreover, ASH inherits the convergence guarantees of gradient descent methods and their flexibility in handling general regularization functions.

We show that ASH outperforms the state-of-the-art algorithms in a number of sparse optimization problems. In particular, ASH exhibits the state-of-the-art convergence in solving the well-studied LASSO problem.
References


Proceedings of the 26th Annual International Conference on Machine Learning (pp. 337–344).: ACM.


[34] Han, S., Mao, H., & Dally, W. J. (2016). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. International Conference on Learning Representations.


83


