The Unreasonable Effectiveness of Machine Learning

A Brief Survey

Sachin Padmanabhan   Vihan Lakshman   Deeksha Goyal

March 2017

Contents

1 Introduction

Since the study of computer algorithms was formalized into a rigorous discipline, worst-case analysis has remained the dominant paradigm for measuring the performance of a computational procedure. And, without a doubt, worst-case analysis has been wildly successful in inspiring the development of new algorithmic ideas and providing a rigorous basis to assess the performance of algorithms relative to each other.

However, worst-case analysis also harbors significant drawbacks. Often, it reports overly pessimistic outlooks on the performance of algorithms since real-word inputs are rarely adversarially generated and don't typically exhibit a "Murphy's Law"-type behavior. In addition, worst-case analysis leaves a large gap between theory and practice. In numerous instances, theoretical computer science lacks the sufficient mathematical framework to explain why the algorithms that deliver the best empirical performance succeed. Conversely, worst-case analysis also offers very little advice to practitioners on how to choose an algorithm to meet the demands of a specific application domain.

Perhaps no field better illustrates this wedge between theory and practice driven by worst-case analysis than machine learning. In many cases, worst-case analysis labels machine learning problems as **NP**-Hard, non-convex, or otherwise intractable while relatively simple heuristics often return outstanding results in practice. With this discrepancy between the tremendous success of machine learning algorithms in revolutionizing the modern world and the conventional theory suggesting otherwise, a major challenge in the modern study of computation is to develop relevant frameworks to explain the unreasonable effectiveness of machine learning.

Results in this burgeoning field of looking beyond worst case analysis to study machine learning algorithm have accelerated considerably in the past five years. In this paper, we survey several of these spectacular results that provide theoretical explanations of why certain bread-and-butter machine learning algorithms perform so well.

We will begin this survey by providing a brief overview of machine learning, particularly supervised learning, as well as convex optimization. It is our hope that this introduction will serve as a valuable pedagogical tool for readers less familiar with the field as well as an opportunity to lay the foundation for our subsequent discussions, which those well-versed in the subject might also find useful.

After the preliminaries, we will discuss a number of major results in the field, surveying publications in the literature all released within the past four years. We will begin with a survey of the result of Hardt, Recht, and Singer in 2015 [HRS15], showing that models trained via stochastic gradient descent (SGD) – arguably the most popular optimization algorithm in machine learning – achieve arbitrarily low generalization error, a result illuminating both in its use of convex analysis and in its ability to theoretically validate one of the most successful algorithms in practice. In our survey, we fill in the details in several of the proofs of the main result.

Next, we will study the famous machine learning heuristic of alternating minimization, which is remarkable both in the clean simplicity of the idea and its remarkable effectiveness in practice. We will three major papers explaining the empirical success of alternating minimization, using two classic machine learning problems, matrix completion and dictionary learning, as our case studies.

Finally, we will end by discussing results discussing algorithms for topic modeling, an unsupervised learning task of great practical interest. In particular, we will survey results exhibiting conditions under which algorithms for topic modeling–an NP-Hard problem–can exactly recover the solution.

2 The Stability of Stochastic Gradient Descent

In this chapter, we will motivate and explain the results in Hardt, Recht, and Singer's 2015 paper on the stability of the stochastic gradient descent method [HRS15].

2.1 The Supervised Learning Setting

In this section, we will consider only the problem of supervised learning. In a supervised learning task, we are confronted with an unknown distribution $\mathcal{D}$ over some space Z. We are given a dataset of n samples $S = (z_1, \ldots, z_n)$, with each sample drawn independently from $\mathcal{D}$. The goal in supervised learning is to compute a model w that is able to "predict" future, unseen examples drawn from $\mathcal{D}$. In order to formalize this, we posit a *loss function* f such that $f(w; z)$ is the *loss* of a model w on the example z. There are many ways to construct a loss function but the key idea is that loss is a bad thing – it implies that the model did not perform well enough on the example z. We now formalize our notion of predicting unseen examples from the distribution $\mathcal{D}$ by defining the *population risk* of a model w as its expected loss on an example in $\mathcal{D}$; that is,

$$R[w] = \mathbf{E}_{z\sim\mathcal{D}} f(w; z).$$

We aim to compute a model with as low population risk as possible.

Unfortunately, we don't know the distribution $\mathcal{D}$, so we cannot calculate $R[w]$. Therefore, we compute an unbiased estimator for $R[w]$ by averaging the loss our model w obtains on examples in the sample set S. The *empirical risk* of w on S is

$$R_S[w] = \frac{1}{n}\sum_{i=1}^{n} f(w; z_i).$$

Instead of directly aiming to minimize the population risk, we aim to compute a model that minimizes the empirical risk on our data set S, which leads to the class of algorithms known as empirical risk minimizers.

Now, we define the generalization error of a model w as

$$R_S[w] - R[w].$$

Clearly, we should aim to compute a model with low generalization error since this implies that the performance obtained on the training data closely mirrors the performance we expect to see on future examples. Since the learning algorithm we consider will be randomized, we will in most cases work with the expected generalization error. In this case, the randomized algorithm A takes as input a sample dataset S and produces the model w. Thus, the expected generalization error attained is

$$\mathbf{E}_S\mathbf{E}_A[R_S[A(S)] - R[A(S)]],$$

where the expectation is over both the random choice of dataset S and the internal random coin flips of the algorithm A.

2.2 Solving Supervised Learning Problems

The first main concern in a supervised learning task is to choose an appropriate loss function f. Until recently, choosing f was incredibly domain-specific and involved domain experts designing hand-crafted features that represented domain-specific knowledge of the problem at hand. Nowadays, with the advent of neural networks and deep learning, practitioners of machine learning harness massive computation power (in many cases, clusters of hundreds of GPUs) to design models with billions of parameters that learn feature representations from the data themselves.

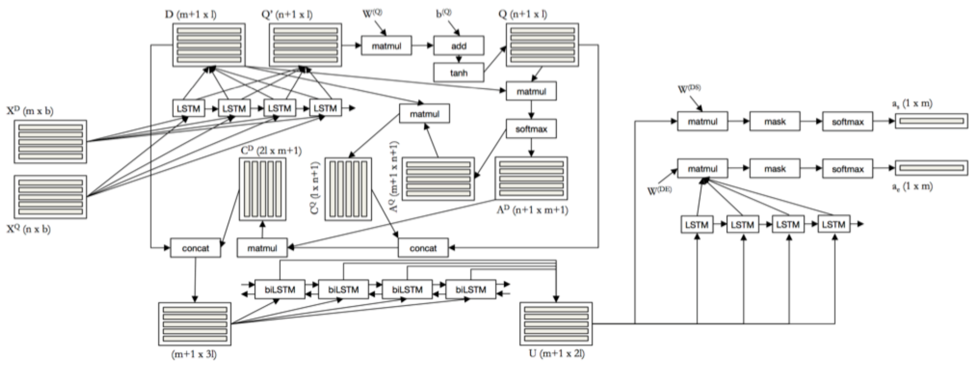

Figure 1: A neural network architecture used in natural language processing.

After the loss function f is chosen, the supervised learning task is to select the model w that minimizes the empirical risk on the entire dataset

$$\frac{1}{n}\sum_{i=1}^{n} f(w; z_i).$$

Thus, after choosing the loss function, the supervised learning task reduces to a continuous optimization problem; namely, that of unconstrained minimization.

In most classical machine learning methods, such as logistic regression, linear regression, and support vector machines, the loss function that is specified for each training example is convex. The aggregate loss function is a nonnegative weighted sum of convex functions, and is thus convex.

In mathematical optimization, convexity is almost synonymous with tractability. Indeed, solving most convex minimization problems, including all the ones mentioned above, is extremely efficient, in theory and in practice. In theory, convex optimization problems can be solved to any required degree of precision in polynomial time using the ellipsoid method. In practice, many superior methods exist, such as gradient methods, second-order methods, and interior-point methods. As a result, convex optimization is often regarded as a black box. Once a problem has been cast as a convex optimization problem, it can be solved blazingly quickly with an off-the-shelf solver. Convex optimization is such an important primitive that large corporations like Google and Facebook invest considerable resources into developing efficient, distributed solvers for massive convex optimization problems that store different parts of the solution on different machines.

2.3 Convex Optimization

We first give the definition of a convex function.

Definition 2.1. *A differentiable function $f : \Omega \to \mathbf{R}$ is* convex *if for all $x, y \in \Omega$,*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x).$$

Alternatively, we can present a characterization based on Jensen's inequality.

Definition 2.2. *A function $f : \Omega \to \mathbf{R}$ is* convex *if for all $x, y \in \Omega$ and $\theta \in [0, 1]$,*

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta) f(y).$$

The salient property of convex functions that makes minimizing them efficient is their "bowl" shape. In particular, any *local minimum* of a convex function is also a *global minimum.*

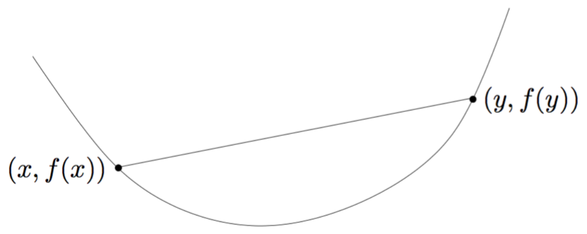

Figure 2: Graph of a convex function. The chord between any two points on the graph lies above the graph.

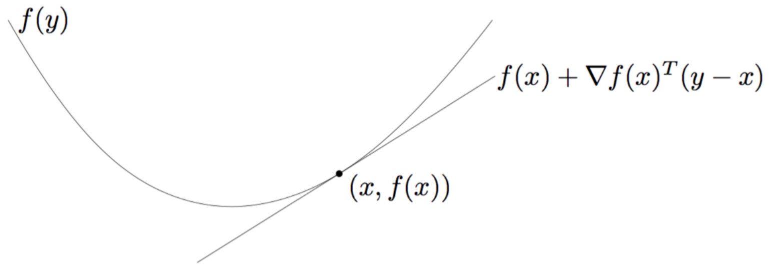

Figure 3: If f is convex and differentiable, then $f(x) + \nabla f(x)^\top (y - x) \leq f(y)$ for all $x, y \in \mathbf{dom}\, f$.

Hence, we can easily minimize convex functions by following our nose. A simple algorithm is to choose an arbitrary starting point in Ω, compute the direction in which the function is decreasing the fastest (given by the negative of the gradient), take a step in that direction, and then repeat. This is known as the *gradient descent method* More formally, we choose a step size $\alpha \geq 0$ and use the following *gradient update rule* repeatedly.

Definition 2.3. *For a function $f : \Omega \to \mathbf{R}$ and step size $\alpha \geq 0$, the* gradient update rule *$G_{f,\alpha}$ is*

$$G_{f,\alpha}(w) = w - \alpha \nabla f(w).$$

Thus, gradient descent corresponds to repeated application of the update $w_{t+1} = G_{f,\alpha}(w_t)$. This method is guaranteed to converge to the global optimum of f, but for very large datasets, computing the gradient in the gradient update rule can be very computationally expensive. Thus, in an effort to improve the computational efficiency of the gradient method, instead of computing the gradient of f on the dataset as a whole, we compute the gradient of f on a randomly chosen training example, and immediately perform the update. This is known as the *stochastic gradient descent method.* More formally, with *learning rate* (i.e. step size) $\alpha_t \geq 0$, and randomly chosen example at time t z_{it}, the *stochastic gradient update* is

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t; z_{it}).$$

Note that this update rule corresponds to the usual gradient update rule on f parameterized with a fixed example z; that is, the function $f(\cdot; z)$.

Stochastic gradient descent (SGD) is a widely used method to optimize convex functions and often converges very quickly (much faster than gradient descent). Moreover, SGD has been shown theoretically and empirically to have many desirable qualities such as small memory footprint and robustness to noise. Indeed, SGD has been shown to be nearly optimal for empirical risk minimization of convex loss functions [NJLS09, NY78, NYD83]. These results have recently been extended to yield tighter bounds and guarantees [HKKA06, HK14, FGKS15]. However, SGD has proven to be applicable in more situations than simply convex optimization.

2.4 Non-Convex Optimization in Practice

In today's world, machine learning is one of the most prominent buzzwords and nothing captures its hype more than the popularity of "deep learning." Neural network models found in practice today commonly have billions of parameters and the resulting objective functions are highly non-convex. Nevertheless, pretty much miraculously, off-the-shelf convex optimization routines such as SGD and other popular methods such as AdaGrad [DHS11] and Adam [KB14], combined with several heuristic tricks, optimize these functions extremely well [KSH12, SLJ$^+$15]. Indeed, in most domains such as computer vision and natural language processing, neural network models vastly outperform pre-existing classical models with convex objective functions. It is still a very much open research question to develop a theory around why these models perform so well and why off-the-shelf *convex* optimization solvers optimize these enormous *non-convex* models so well.

Indeed, it might even be the case that convex optimization methods work so well on these non-convex optimization problems precisely because they are heuristic. First of all, in machine learning, the loss function itself is a proxy and "better" models in practice need not correspond to lower loss function values. Moreover, it may well be the case that if we were to find a global minimum for these large non-convex objectives, it would just result in a model that grossly overfits the data.

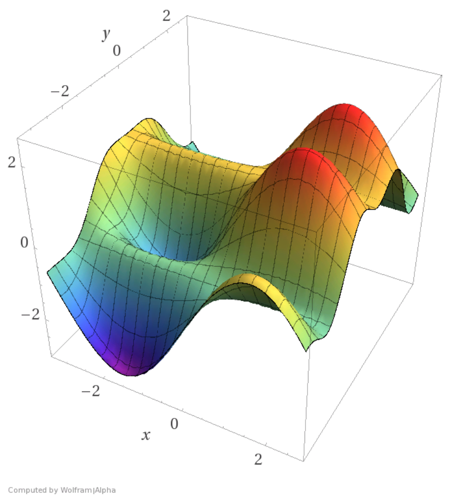

Figure 4: A non-convex function can have local minima that are not global minima.

2.5 The Main Result

Because of the unreasonable effectiveness of convex optimization routines on patently non-convex optimization problems, it's necessary to ask the question: why do these heuristic methods, like the stochastic gradient method, work? This question was partially answered by Hardt, Recht, and Singer at Google Research, an organization that generates its fair share of non-convex objective functions through its extensive work in deep learning. At a very high level, they established the following theorem.

Theorem 2.1. *Any model trained using SGD as the optimization algorithm in a "reasonable" amount of time attains "small" generalization error.*

More concretely, the authors bound the generalization error that a model trained with SGD attains as a function of the number of iterations that SGD takes to converge to a low error threshold. As noted previously, many bounds and guarantees exist for SGD in the convex setting. However, these are largely limited to analyzing one pass over the data. This paper does not make that restriction.

This result is significant both in theory and in practice. Training times for the billion-parameter scale models that are common in practice today are on the order of days or weeks and so models that train faster are preferred to those that take longer to train. This result justifies using these models that train faster and also explains the strong performance of models trained with SGD in practice. Moreover, since training times are currently so excruciatingly long, it is an active area of research and development to construct models and methods that decrease training time. This result shows that aiming to reduce the training time for a model is indeed an appropriate goal in that models that train faster will achieve better generalization error. Hence, the result inspires the development of models that train faster with SGD, suggesting that these models will also generalize well.

Recall the three main goals of the analysis of algorithms:

1. **Performance Prediction**: Explain or predict the empirical performance of algorithms.
2. **Identify Optimal Algorithms**: Rank different algorithms according to their performance, and ideally single out one algorithm as "optimal."
3. **Design New Algorithms**: Guide the development of new algorithms.

With respect to these goals, the main result in this paper makes significant contributions to the analysis of SGD, machine learning, and non-convex optimization. In particular, it makes a major contribution to the first goal of Performance Prediction in that it explains why models, even extremely complex ones, trained by SGD, perform extremely well on real-world tasks. It also reaffirms the practice of attempting to minimize training times. Furthermore, the result also makes significant progress on the third goal of inspiring the design of new "algorithms." In this case, the authors provide a metric with which practitioners can judge the expected generalization of a complex model: the number of iterations SGD takes until convergence. Thus, practitioners can use this as a guideline to design new models which will train quickly with SGD and thus attain good generalization.

2.6 Uniform Stability

The main idea of the argument is a motif that comes up frequently in beyond worst-case analysis of algorithms: we first define a certain notion of stability and show that if an algorithm satisfies this particular notion of stability, then it has the desired property. Then, we proceed to show that the protagonist algorithm, in this case stochastic gradient descent, is stable, showing that it has the desired property. This method exactly parallels the perturbation stability arguments used to analyze the Single-Link++ algorithm for clustering and linear programming and semidefinite programming relaxations for the minimum cut and maximum cut problems [MM16, MMV14].

Showing that SGD has this notion of stability and that this notion of stability implies good generalization also improves the result's score on the Performance Prediction goal of algorithmic analysis since it inspires researchers to show that other extremely popular optimization heuristics such as AdaGrad or Adam have a similar stability property and thus have similar generalization bounds.

Informally, the notion of stability used by the authors is that on any two training sets differing in at most one point, the learned models are almost the same. More specifically, they use a slightly adapted definition of *uniform stability* for a randomized algorithm, initially introduced by Bousquet and Elisseeff [BE02]

Definition 2.4. *A randomized algorithm A is* ϵ-uniformly stable *if for all datasets $S, S' \in Z^n$ such that S and S' differ by at most one example,*

$$\sup_x \mathbf{E}_A[f(A(S); z) - f(A(S'); z)] \leq \epsilon,$$

where the expectation is over the internal random coin flips of the algorithm A. Additionally, let ϵ_{stab} be the smallest ϵ for which the above inequality holds.

2.7 Uniform Stability Implies Generalization in Expectation

We now show that the notion of uniform stability of a learning algorithm introduced above implies good performance (i.e. generalization) of the learning algorithm. More precisely, we will show that if a learning algorithm A is ϵ-uniformly stable, then the magnitude of the expected generalization error (over the choice of the training set and internal randomness of A) is at most ϵ. Hence, ϵ_{stab} gives us the sharpest bound we can get with this stability analysis on the expected generalization error of A.

Theorem 2.2. *If A is ϵ-uniformly stable, then*

$$|\mathbf{E}_S \mathbf{E}_A[R_S[A(S)] - R[A(S)]]| \leq \epsilon.$$

Proof. Let $S = (z_1, \ldots, z_n)$ and $S' = (z'_1, \ldots, z'_n)$ be two datasets drawn independently at random from the distribution $\mathcal{D}$. Define a new dataset $S^{(i)}$ to be the dataset that is identical to S except in the i'th index, where z_i is replaced with z'_i. That is, $S^{(i)} = (z_1, z_2, \ldots, z_{i-1}, z'_i, z_{i+1}, \ldots, z_{n-1}, z_n)$. Then, using linearity of expectation and the fact that S and S' are both drawn independently at random from the same distribution and hence $\{S^{(i)}\}_{i=1}^n$ ranges over the same distribution:

$$\begin{aligned}
\mathbf{E}_S \mathbf{E}_A[R_S[A(S)]] &= \mathbf{E}_S \mathbf{E}_A \left[\frac{1}{n} \sum_{i=1}^n f(A(S); z_i) \right] \\
&= \mathbf{E}_S \mathbf{E}_{S'} \mathbf{E}_A \left[\frac{1}{n} \sum_{i=1}^n f(A(S^{(i)}); z'_i) \right] \\
&= \mathbf{E}_S \mathbf{E}_{S'} \mathbf{E}_A \left[\frac{1}{n} \sum_{i=1}^n f(A(S); z'_i) \right] + \delta,
\end{aligned}$$

where

$$\delta = \mathbf{E}_S \mathbf{E}_{S'} \mathbf{E}_A \left[\frac{1}{n} \sum_{i=1}^n f(A(S^{(i)}); z'_i) - \frac{1}{n} \sum_{i=1}^n f(A(S); z'_i) \right].$$

With this δ, we have that the term in the expectation is precisely the population risk of $A(S)$; that is, we have that $\mathbf{E}_S \mathbf{E}_A[R_S[A(S)]] = \mathbf{E}_S \mathbf{E}_A[R[A(S)] + \delta$. Now, we can note that pairs $(S, S^{(i)})$ range over all pairs of datasets (S, S') that differ by only one sample, so we can bound $|\delta|$ by computing its value for the worst-case datasets S and S' and example z; that is,

$$|\delta| \leq \sup_{S,S',z} \mathbf{E}_A[f(A(S); z) - f(A(S'); z)].$$

But, since A is ϵ-uniformly stable, $\sup_{S,S',z} \mathbf{E}_A[f(A(S); z) - f(A(S'); z)] \leq \epsilon$. Thus,

$$|\delta| = |\mathbf{E}_S\mathbf{E}_A[R_S[A(S)]] - \mathbf{E}_S\mathbf{E}_A[R[A(S)]]| = \mathbf{E}_A\mathbf{E}_A[R_S[A(S)] - R[A(S)]] \leq \epsilon,$$

as desired. □

Thus, it only remains to show that SGD is ϵ-uniformly stable for some ϵ. This will show that the generalization error attained by models trained with SGD is bounded by ϵ. Indeed, ϵ will be a function of the number of iterations SGD runs for.

2.8 Analysis Tool: Update Rules

Recall the gradient update rule $G_{f,\alpha}(w) = w - \alpha\nabla f(w)$, with $\alpha \geq 0$, which updates the current model by stepping the current parameters in the direction antipodal to the current gradient. In this section, we will build up general tools that will allow us to bound the results of update rules. Consider a general update rule $G : \Omega \to \Omega$. We first give some basic definitions.

Definition 2.5. *An update rule $G : \Omega \to \Omega$ is η-expansive if*

$$\sup_{v,w\in\Omega} \frac{\|G(v) - G(w)\|}{\|v - w\|} \leq \eta.$$

Definition 2.6. *An update rule $G : \Omega \to \Omega$ is σ-bounded if*

$$\sup_{w\in\Omega} \|w - G(w)\| \leq \sigma.$$

Armed with these definitions, we can proceed to utilize them to bound the extent to which two sequences of updates to a model can diverge, as a function of the expansiveness and boundedness of the constituent updates. We start with the setup.

Let the first update sequence be $G_1, G_2, \ldots, G_T$ and the second be $G'_1, G'_2, \ldots, G'_T$. Suppose the two sequences are initialized with the same starting model w. Let w_t be the model parameters of the first update sequence at any time t and similarly let w'_t be the model parameters of the second update sequence throughout time. Formally, we let $w_0 = w'_0$ and define $w_{t+1} = G_t(w_t)$ and $w'_{t+1} = G'_t(w'_t)$. Finally, we let $\delta_t = \|w_t - w'_t\|$ be the extent to which the two learned models differ at time t. Clearly, $\delta_0 = 0$.

Hence, we can bound the extent to which the two models diverge at the end of training by bounding δ_T. We will do this by establishing a recurrence relation that shows that $\delta t + 1$ is not too much larger than δ_t given appropriate expansiveness or boundedness of the constituent updates. Formally, we show the following lemma.

Lemma 2.7. *For two sets of update rules $G_1, \dots, G_T$, $G'_1, \dots, G'_T$ with w_t, w'_t, and δ_t defined as above, the following recurrence relation holds for all $t \in \{0, \dots, T-1\}$:*

$$\delta_{t+1} \leq \begin{cases} \eta\delta_t & G_t = G'_t \text{ is } \eta\text{-expansive} \\ \min(\eta, 1)\delta_t + 2\sigma & G_t \text{ and } G'_t \text{ are } \sigma\text{-bounded and } G_t \text{ is } \eta\text{-expansive} \end{cases}$$

Proof. For the first bound, suppose $G_t = G'_t$ is η-expansive. Then, let $G = G_t = G'_t$. Applying the η-expansiveness property to G at the points w_t and w'_t gives that

$$\frac{\|G(w_t) - G(w'_t)\|}{\|w_t - w'_t\|} \leq \eta.$$

This directly implies that $\|w_{t+1} - w'_{t+1}\| \leq \eta\|w_t - w'_t\|$; that is, $\delta_{t+1} \leq \eta\delta_t$.

Now, for the second bound, suppose that G_t and G'_t are σ-bounded and that G_t is η-expansive. First, we use the triangle inequality to write

$$\begin{aligned} \delta_{t+1} &= \|G_t(w_t) - G'_(w'_t)\| \\ &= \|G_t(w_t) - G'_t(w'_t) - w_t + w'_t + (w_t - w'_t)\| \\ &\leq \|G_t(w_t) - w_t + w'_t - G'_t(w'_t)\| + \|w_t - w'_t\| \\ &\leq \|G_t(w_t) - w_t\| + \|w'_t - G'_t(w'_t)\| + \|w_t - w'_t\| \end{aligned}$$

Now, since G_t and G'_t are σ-bounded, $\|G_t(w_t) - w_t\| \leq \sigma$ and $\|w'_t - G'_t(w'_t)\| \leq \sigma$. Thus,

$$\begin{aligned} \delta_{t+1} &= \|G_t(w_t) - G'_(w'_t)\| \\ &\leq \|G_t(w_t) - w_t\| + \|w'_t - G'_t(w'_t)\| + \|w_t - w'_t\| \\ &\leq \delta_t + 2\sigma. \end{aligned}$$

We can also bound δ_{t+1} in a different way, giving a different bound. The process is along the same lines, using the triangle inequality and the σ-boundedness of G_t and G'_t, but in addition using the η-boundedness of G_t in the last step.

$$\begin{aligned} \delta_{t+1} &= \|G_t(w_t) - G'_(w'_t)\| \\ &= \|G_t(w_t) - G_t(w'_t) + G_t(w'_t) - G'_(w'_t)\| \\ &\leq \|G_t(w_t) - G_t(w'_t)\| + \|G_t(w'_t) - G'_(w'_t)\| \\ &= \|G_t(w_t) - G_t(w'_t)\| + \|G_t(w'_t) - w'_t + w'_t - G'_(w'_t)\| \\ &\leq \|G_t(w_t) - G_t(w'_t)\| + \|G_t(w'_t) - w'_t\| + \|w'_t - G'_(w'_t)\| \\ &\leq \eta\|w_t - w'_t\| + 2\sigma \\ &= \eta\delta_t + 2\sigma. \end{aligned}$$

□

2.9 Analysis Tool: Smoothness

In the previous section, we gave basic definitions that gave bounds on the output of update rules. In this section, we provide standard definitions of two different "smoothness" properties that will allow us to derive bounds on the two properties in the previous section (expansiveness and boundedness) specifically for the gradient update rule.

Definition 2.8. *A function f is L-Lipschitz if for all points $x \in \mathbf{dom}\, f$, $\|\nabla f(x)\| \leq L$. As a direct consequence, for any two points $u, v \in \mathbf{dom}\, f$,*

$$|f(u) - f(v)| \leq L\|u - v\|.$$

Definition 2.9. *A function f is β-smooth if for all $u, v \in \mathbf{dom}\, f$,*

$$\|\nabla f(u) - \nabla f(v)\| \leq \beta\|u - v\|.$$

Notice that just as the Lipschitz condition is assures that the *value* of the function f does not change too fast (with the rate controlled by L), the smoothness condition assures that the *derivative* of f does not change too fast (with the rate controlled by β).

2.10 Applying the Tools: Expansiveness of Gradient Updates

In this section, we will see that assuring that f is Lipschitz and smooth results in bounds on the two expansion properties of the gradient update $G_{f,\alpha}$ introduced in Section 2.8: boundedness and expansiveness, respectively. We present these bounds as two lemmas. The first lemma, which shows that if f is Lipschitz, then the gradient update $G_{f,\alpha}$ is bounded, is a simple application of the Lipschitz property.

Lemma 2.10. *If f is L-Lipschitz, then the gradient update $G_{f,\alpha}$ is (αL)-bounded.*

Proof. Using the definition of the gradient update rule,

$$\begin{aligned}
\|w - G_{f,\alpha}(w)\| &= \|w - (w - \alpha\nabla f(w))\| \\
&= \|\alpha\nabla f(w)\| \\
&= \alpha\|\nabla f(w)\| \\
&\leq \alpha L,
\end{aligned}$$

where the inequality follows because f is L-Lipschitz. □

The second lemma, which categorizes the expansiveness of $G_{f,\alpha}$ for a β-smooth function f is a relatively simple technical lemma which can be found in several resources, such as [Nes13].

Lemma 2.11. *If f is β-smooth, then the gradient update $G_{f,\alpha}$ is $(1 + \alpha\beta)$-expansive. Moreover, if f is convex, then for any $\alpha \leq 2/\beta$, $G_{f,\alpha}$ is 1-expansive.*

Proof. We first show that $G_{f,\alpha}$ is $(1 + \alpha\beta)$-expansive. Using the definition of the gradient update rule,

$$\begin{aligned}
\|G_{f,\alpha}(v) - G_{f,\alpha}(w)\| &= \|(v - \alpha\nabla f(v)) - (w - \alpha\nabla f(w))\| \\
&= \|(v - w) - \alpha(\nabla f(v) - \nabla f(w))\| \\
&\leq \|v - w\| + \alpha\|\nabla f(v) - \nabla f(w)\|,
\end{aligned}$$

where the inequality follows from the triangle inequality. Now, since f is β-smooth,

$$\begin{aligned}\|G_{f,\alpha}(v) - G_{f,\alpha}(w)\| &\leq \|v - w\| + \alpha\|\nabla f(v) - \nabla f(w)\| \\ &\leq \|v - w\| + \alpha\beta\|v - w\| \\ &= (1 + \alpha\beta)\|v - w\|,\end{aligned}$$

showing that $G_{f,\alpha}$ is indeed $(1 + \alpha\beta)$-expansive.

To show the second part of the lemma, we use a fact from convex analysis. Namely, if f is convex and β-smooth, then ∇f co-coercive; that is,

$$(\nabla f(v) - \nabla f(w))^\top (v - w) \geq \frac{1}{\beta}\|\nabla f(v) - \nabla f(w)\|^2.$$

The proof of this fact comes from the fact that the β-smoothness of f implies that the function $g(x) = (\beta/2)x^\top x - f(x)$ is convex. This in turn implies that ∇f is co-coercive.

Now, again using the definition of the update rule,

$$\begin{aligned}\|G_{f,\alpha}(v) - G_{f,\alpha}(w)\|^2 &= \|(v - \alpha\nabla f(v)) - (w - \alpha\nabla f(w))\|^2 \\ &= \|(v - w) - \alpha(\nabla f(v) - \nabla f(w))\|^2 \\ &= \|v - w\|^2 - 2\alpha(\nabla f(v) - \nabla f(w))^\top(v - w) + \alpha^2\|\nabla f(v) - \nabla f(w)\|^2.\end{aligned}$$

Applying the co-coercivity of ∇f,

$$\begin{aligned}\|G_{f,\alpha}(v) - G_{f,\alpha}(w)\|^2 &= \|v - w\|^2 - 2\alpha(\nabla f(v) - \nabla f(w))^\top(v - w) + \alpha^2\|\nabla f(v) - \nabla f(w)\|^2 \\ &\leq \|v - w\|^2 - \frac{2\alpha}{\beta}\|\nabla f(v) - \nabla f(w)\|^2 + \alpha^2\|\nabla f(v) - \nabla f(w)\|^2 \\ &= \|v - w\|^2 - \left(\frac{2\alpha}{\beta} - \alpha^2\right)\|\nabla f(v) - \nabla f(w)\|^2 \\ &\leq \|v - w\|^2,\end{aligned}$$

where the last inequality holds since $\alpha \leq 2/\beta$, causing the second term to be nonnegative. This implies that $\|G_{f,\alpha}(v) - G_{f,\alpha}(w)\| \leq \|v - w\|$, showing that the gradient update $G_{f,\alpha}$ is indeed 1-expansive. □

2.11 Convex Loss Functions

We are now ready to state and prove the stability result for stochastic gradient descent on convex objective functions.

Theorem 2.3. *Suppose that for every example z, the loss function on that example $f(\cdot; z)$ is convex, L-Lipschitz, and β-smooth. Moreover, suppose that we run SGD with step sizes $\alpha_t \leq 2/\beta$ for T steps. Then, SGD is uniformly stable with*

$$\epsilon_{stab} \leq \frac{2L^2}{n}\sum_{t=1}^{T}\alpha_t.$$

Proof. We start with the high-level idea. Let S and S' be as in the setup: two data sets with n training examples each, which differ at exactly one index. Let the gradient updates obtained by running SGD on S be $G_1, \dots, G_T$ and let the updates obtained on S' be $G'_1, \dots, G'_T$. Furthermore, let the learned models at each timestep t be w_t and w'_t.

At any timestep t we can fall into one of two cases. In the first case, SGD chooses the index of a training example that is identical in S and S' It turns out that since the two learned models at this time w_t and w'_t can differ, the gradients can differ and thus the gradient updates may not be the same. Nevertheless, we can bound the increase in δ by using the machinery developed around convexity and smoothness earlier.

In the second case, which happens with probability $1/n$, SGD chooses the index of the training example that differs between S and S'. In this case, we will bound the increase in δ by the sum of the norm of the two gradient updates.

Now, we give the proof. Consider the first case, in which SGD selects an index in which the training sets S and S' agree, which happens with probability $1 - 1/n$. In this case, the update rules are precisely the same; that is, $G_t = G'_t$. Thus, by Lemma 2.11, which states that $G_t = G'_t$ is 1-expansive, together with Lemma 2.11, we have that $\delta_{t+1} \leq \delta_t$.

Now, consider the second case, in which SGD selects the index in which the training sets S and S' disagree. This case occurs with probability $1/n$. In this case, we can bondδ_{t+1} with Lemma 2.7 by using the fact that G_t and G'_t are $\alpha_t L$-bounded, by Lemma 2.10. Now, we use the recurrence lemma 2.7 and linearity of expectation to find that

$$\begin{aligned}\mathbf{E}[\delta_{t+1}] &\leq \left(1 - \frac{1}{n}\right)\mathbf{E}[\delta_t] + \left(\frac{1}{n}\right)(\mathbf{E}[\delta_t] + 2\alpha_t L) \\ &= \mathbf{E}[\delta_t] + \frac{2\alpha_t L}{n}.\end{aligned}$$

Thus, noting that $\delta_0 = 0$,

$$\mathbf{E}[\delta_T] \leq \sum_{t=1}^{T} \frac{2\alpha_t L}{n} = \frac{2L}{n}\sum_{t=1}^{T} \alpha_t.$$

Finally, fix an arbitrary example z. By assumption, $f(\cdot; z)$ is L-Lipschitz. Thus,

$$\mathbf{E}[|f(w_T; z) - f(w'_T; z)|] \leq \mathbf{E}[L\|w_T - w'_T\|] = L\mathbf{E}[\delta_T].$$

By the previous result,

$$\mathbf{E}[|f(w_T; z) - f(w'_T; z)|] \leq L \cdot \frac{2L}{n}\sum_{t=1}^{T} \alpha_t = \frac{2L^2}{n}\sum_{t=1}^{T} \alpha_t.$$

Since this bound holds for any arbitrary z and the datasets S and S' were chosen arbitrarily, this yields the uniform stability bound. $\square$

2.12 Non-Convex Loss Functions

With non-convex loss functions, we no longer have the nice property that our gradient updates $G_{f,\alpha}$ are nonexpansive. However, we do know that they are not *too* expansive; that is, $G_{f,\alpha}$ is $(1+\alpha\beta)$-expansive. However, this enough is not alone to give the desired uniform stability bound. Nevertheless, the authors were able to give a uniform stability bound on SGD on a non-convex loss function.

Theorem 2.4. *Suppose that for every example z, the loss function on that example $f(\cdot; z)$ is L-Lipschitz and β-smooth, and moreover lies within the interval $[0,1]$. If we run SGD for T steps with monotonically increasing step sizes $\alpha_t \leq c/t$, then SGD is uniformly stable with*

$$\epsilon_{stab} \leq \frac{1+1/(\beta c)}{n-1}\left(2cL^2T^{\beta c}\right)^{1/(\beta c+1)}.$$

We will not provide the proof here as it is technical and follows from the same principles as in the convex case, with additional algebraic manipulations. Instead, we provide the proof idea and motivation.

The clever realization by the authors is that we can divide the execution of SGD into two phases. In Phase I, SGD does not ever encounter the one training example on which S and S' differ. Phase II starts immediately once SGD encounters the differing training example. Notice that throughout Phase I, the models learned on the two training sets, w and w' are precisely the same; thus, they only start to diverge in Phase II. The authors argue that Phase I must last a sufficiently long time, showing that by the time Phase II starts, the step size has already decayed enough as to bound the difference in the models throughout Phase II. In particular, let the random variable I denote the index of the first time step in which SGD picks the index at which S and S' differ. Then, for any $t_0 \in \{0, 1, \ldots, n\}$, we use the union bound to show that

$$\begin{aligned}\Pr[I \leq t_0] &\leq \sum_{t=1}^{t_0} \Pr[I = t]\\ &= \sum_{t=1}^{t_0} \frac{1}{n}\\ &= \frac{t_0}{n}.\end{aligned}$$

2.13 Stability-Inducing Operations in Practice

Training a large machine learning model in practice involves numerous tricks to improve the training time and performance of the model. Many of these methods have been widely empirically verified to improve training time, generalization error, or both. Most notably, it is common in practice to use weight decay and/or dropout. The authors showed that each of these (and other common performance-improving methods) improve the stability property of SGD, thereby bolstering the case for their use in practice.

We illustrate the case with weight decay, as introduced by [KH91]. In weight decay, we use a modified gradient update. We let the gradient update with weight decay μ be

$$G_{f,\mu,\alpha}(w) = (1-\alpha\mu)w - \alpha\nabla f(w).$$

Then, we have the following result:

Claim 2.12. *If f is β-smooth, then $G_{f,\mu,\alpha}$ is $(1+\alpha(\beta-\mu))$-expansive. (Notice that this is an improvement over $1+\alpha\beta$!)*

Proof. Using the new gradient update rule, for any $v, w \in \Omega$,

$$\begin{aligned}\|G_{f,\mu,\alpha}(v) - G_{f,\mu,\alpha}(w)\| &= \|((1-\alpha\mu)v - \alpha\nabla f(v)) - ((1-\alpha\mu)w - \alpha\nabla f(w))\| \\ &= \|(1-\alpha\mu)(v-w) - \alpha(\nabla f(v) - \nabla f(w))\| \\ &\leq (1-\alpha\mu)\|v-w\| + \alpha\|\nabla f(v) - \nabla f(w)\|,\end{aligned}$$

where the last inequality follows because of the triangle inequality. Now, since f is β-smooth, $\|\nabla f(v) - \nabla f(w)\| \leq \beta\|u-v\|$. Thus, since $\alpha \geq 0$,

$$\begin{aligned}\|G_{f,\mu,\alpha}(v) - G_{f,\mu,\alpha}(w)\| &\leq (1-\alpha\mu)\|v-w\| + \alpha\|\nabla f(v) - \nabla f(w)\| \\ &\leq (1-\alpha\mu)\|v-w\| + \alpha\beta\|v-w\| \\ &= (1+\alpha(\beta-\mu))\|v-w\|,\end{aligned}$$

showing that $G_{f,\mu,\alpha}$ is $(1+\alpha(\beta-\mu))$-expansive. □

Note that adding weight decay can thus greatly improve the stability properties and hence the performance of SGD. For example, if $\mu = \beta$, then the resulting gradient updates are nonexpansive, as in the convex case, even if the objective function is non-convex! Furthermore, if $\mu > \beta$, then the resulting gradient updates are *contractive*, as in the case when the objective function is *strongly* convex.

Dropout, introduced by [SHK$^+$14], is a method used in practice to both improve training times and reduce overfitting for large neural networks. In dropout, a random fraction s of the model weights are set to 0. Under the assumption that f is L-Lipschitz, using a dropout rate of s results in a gradient update that is $(s\alpha L)$-bounded, which is again, better than before, when the gradient update was (αL)-bounded.

2.14 Conclusion

Stability strikes again. We saw that using a notion of stability, we were able to show that this notion implies generalization in expectation. We were then able to show that SGD, the algorithm we care about, satisfies the notion of stability and hence has generalization in expectation.

SGD is a very common algorithm in practice and reliably works very well for many machine learning tasks, including extremely non-convex tasks for which it has no reason to work well.

Thus, this work represents a step forward in understanding *why* the simple heuristic of applying a standard convex optimization routine works so well in optimizing non-convex functions.

But maybe there is something special about SGD. It is well known to have a small memory footprint and be robust against noise. The result in this paper suggests that SGD is exactly what you want when doing machine learning. It achieves fast training, and when it achieves fast training, the model learned has provable guarantees of performing well. Moreover, designing models for which SGD trains quickly can be used as a guideline to design new models which will attain good generalization error.

It has also been shown that SGD has many other intriguing properties that cause it to continue to be widely used in industry. For example, SGD can be run in *parallel* across multiple nodes *without* any locking of shared resources used between processes. This results in massive parallelism and it can be shown that if the gradient updates only modify small portions of the decision variable, then the resulting scheme, Hogwild!, achieves a nearly optimal rate of convergence [RRWN11, SZOR15].

Clearly, there is much more to understand about SGD, and applying convex optimization routines in novel ways to both convex and non-convex optimization problems.

3 Alternating Minimization for Matrix Completion

3.1 Introduction

A major theme throughout this survey is highlighting algorithms and heuristics commonly used to tackle bread-and-butter problems in machine learning that, upon first glance, prompt a reaction along the lines of "Why the heck does that even work?" One classic example of this phenomenon is the *alternating minimization* heuristic to transform non-convex, often intractable, optimization problems involving matrix products into a convex formulation by fixing one matrix, optimizing over the other, and then reversing the roles and proceeding in an iterative fashion. In practice, this heuristic has been employed countless times with stellar results on a variety of ML tasks with the relevant theory only recently catching up in providing a framework of explaining why alternating minimization so often succeeds.

In this chapter and the following one, we study the borderline unbelievable effectiveness of alternating minimization through two case studies. First, we will explore why alternating minimization leads to effective algorithms for a class of machine learning problems that have gained considerable traction in the last decade: matrix completion. Then, in the next section, we will examine the role of alternating minimization in the context of sparse recovery and dictionary learning. In the process, we will not only encounter exciting algorithmic results, but also get a taste of the broad scope of machine learning.

3.2 The Netflix Prize

It's almost impossible to introduce the matrix completion problem without first discussing the ultimate killer app: recommendation systems.

In 2009, Netflix, the online movie streaming service, announced a world-wide open competition [1] to predict consumer ratings of movies based on prior data on user-film ratings with a prize of $\$1,000,000$ in cold hard cash to the most accurate model. The training data provided by Netflix consisted of over 100,000 ratings that 480,000+ users gave to nearly 20,000 movies. The learning task was defined as predicting the ratings that other users would assign to particular films.

We can view this information on users and movie ratings as a large matrix M where the rows represent customers and columns represent movies. The ijth entry of the matrix thus represents the rating that user i gave to film j. We observe that this matrix is largely incomplete since every viewer has not seen every possible movie. Thus, we hope to fill in the missing elements of the matrix and, by extension, recommend movies to users for which we predict a high rating.

[1] `https://en.wikipedia.org/wiki/Netflix_Prize#Problem_and_data_sets`

$$\begin{array}{ccccc|l} \text{Moonlight} & \text{Spotlight} & \text{Birdman} & \text{Argo} & \text{Gladiator} & \\ 3 & 2 & ? & ? & ? & \text{User 1} \\ ? & ? & 4 & ? & 5 & \text{User 2} \\ 1 & ? & ? & 5 & 2 & \text{User 3} \end{array}$$

3.3 Formalizing the Matrix Completion Problem

As we have described the matrix completion problem so far, one might notice that the task is rather uninteresting. Without any sort of additional constraints, we could assign arbitrary values to the unknown entries of M and call it a day. In other words, the problem as it stands is underdetermined. To remedy this, we will impose the additional condition to choose the unknown values so as to minimize the rank of the matrix. In addition to making the problem more interesting, the low-rank objective is well motivated. In the Netflix prize for example, a low rank would correspond to a small number of features which aligns with the reality that only a few predominant factors (e.g. genre, release date, box office sales) influence the rating of a movie. In addition, for the purpose of algorithmic analysis, we will typically study the *exact* matrix completion problem which aims to fill in each of the unknown values of M correctly (assuming a ground-truth fully complete M exists). Another popular variant is the *noisy* matrix completion problem which relaxes the demand on exactly matching M and instead insists that the final matrix XY^T is sufficiently close to M, where we measure "sufficiently close" by examining the Frobenius norm between M and XY^T. In this survey, we will primarily concern ourselves with the former case while saying a few words about analogous results for noisy matrix completion.

3.4 Alternating Minimization

As it turns out, the matrix completion problem as described above is **NP**-Hard. Nevertheless, several highly successful heuristics for the problem have been developed and used in practice with alternating minimization chief among them. In fact, successful algorithms for the Netflix prize utilized some variant of the alternating minimization approach, and the heuristic continues to play a key role in many other matrix completion applications.

Let $M \in \mathbb{R}^{m\times n}$ be the matrix we wish to complete and let S be the subset of entries already known. Fix a target rank k. We will attempt to write M in bi-linear form as the product of two matrices $X \in \mathbb{R}^{m\times k}$ and $Y \in \mathbb{R}^{n\times k}$. That is, we hope to factor M into X and Y so that $M = XY^T$. Since X and Y each have k columns, this effectively decomposes M into two significantly smaller matrices. Now, we can view matrix completion through the lens of an optimization problem, namely one that seeks to minimize the error between X, Y and the known entries of A.

As we alluded to in the preliminary discussion, this optimization problem is non-convex and difficult to solve directly, so we turn to alternating minimization. To start, we can initialize

X and Y (using a standard initialization procedure) so that XY^T is likely at first to be a poor approximation of M. Then, we iteratively refine the values of X and Y by holding one matrix fixed and then optimizing over the other. In other words, at time step t we solve the optimization problem

$$X_t = \arg\min_X \sum_{i,j\in S} \left[M_{ij} - \left(XY_{t-1}^T\right)_{ij}\right]^2$$

where we fix Y from the previous iteration and minimization the squared error between M and XY^T over every known component of M. Then, we finish the t-th iteration by fixing X and optimizing over Y

$$Y_t = \arg\min_Y \sum_{i,j\in S} \left[M_{ij} - \left(X_t Y^T\right)_{ij}\right]^2$$

And that's the entirety of alternating minimization! Since matrix completion is NP-Hard, we don't expect this procedure to allow us to complete M in polynomial time. Nevertheless, we find that alternating minimization often performs extremely well. Moreover, it is a very attractive approach to employ in practice because of its computational efficiency (each update involves solving a simple convex optimization problem) and the fact that it utilizes very little memory (we only need to keep track of the entries of X, Y and S). By comparison, the nuclear norm another well-known matrix completion heuristic, is much more costly since it requires finding the solution to a semidefinite program. This inefficiency makes the nuclear norm approach an unworkable heuristic on very large matrices, which bolsters the case for alternating minimization even further.

3.5 Conditions for Near-Exact Recovery with Alternating Minimization

We now jump to into the technical meat of this chapter and discuss conditions under which we can guarantee that alternating minimization will allow us to exactly recover the target matrix M. Such a theoretical underpinning for alternating minimization was noticeably lacking in the literature until a 2012 breakthrough paper by Jain, Netrapalli, and Sanghavi [JNS13] along with Moritz Hardt's excellent followup paper and exposition of the problem [Har13]. In this section, we will follow Hardt's paper and review one his major result bounding how many elements of the matrix M one needs to see to guarantee near-exact recovery. We will then discuss Hardt's subsequent results at a high level, omitting proofs of the highly technical lemmas favor of summarizing the techniques involved, especially with regards to how one might generalize these ideas to attack similar problems in machine learning.

To set up the problem, we make a slight twist of our definition of the exact matrix completion problem. We still posit the goal of exactly recovering a matrix M of rank k from a subset S of components of M. Now, however, we will introduce a probabilistic element to our model and assume that each entry in M is included in S with probability p. Moreover, we will now assume that M is an $n \times n$ symmetric square matrix with factorization $U\Lambda U^T$. This last assumption is without loss of generality as we can always transform a square matrix into a rectangular matrix via a *dilation* (see Appendix D of the original paper [Har13] for details)

Before we can state the main result of this section, we introduce another definition, coherence. Intuitively, one can view coherence as a "degree of difficulty" for exact recovery of M. Thus, a higher coherence measure indicates more work needs to be done to exactly recover the matrix, and that is exactly what we shall see.

Definition 3.1. *Let U be a matrix. The* coherence *of U, denoted $\mu(U)$ is*

$$\max_{i=1,2,\ldots,n}(n/k)\|e_i^T U\|_2^2$$

where e_i is an n dimensional vector with a 1 in the ith component and zeros everywhere else.

Now we are ready to state one of Hardt's main results regarding the sample complexity required for exact recovery in matrix completion. In this case, sample complexity refers to the number of elements of M one needs to observe in order to achieve this guarantee.

Theorem 3.1. *Given a sample size of $\widetilde{O}(pn^2)$ drawn from $M = U\Lambda U^T$ by including each entry with probability p, there exists an algorithm that outputs with high probability a pair of matrices X and Y such that $||(I - UU^T)X|| \leq \varepsilon$ and $||M - XY^T||_F \leq \varepsilon||M||_F$ if*

$$pn \geq k(k + \log(n/\varepsilon))\mu(U)(||M||_F/\sigma_k)^2$$

where $||A||_F$ denotes the Frobenius norm of the matrix A and σ_k denotes the kth singular value of M.

This is a very exciting result. After spending much of the early part of this chapter lamenting the long-time lack of provable positive results on the performance of alternating minimization, we have such a guarantee – almost. The one caveat is that we do not use alternating minimization in it's pure, unadulterated form. Instead, Hardt devises a clever algorithm based on alternating minimization to achieve the bound. This distinction, though, is rather meaningless as our only goal was to develop some kind of theory that could explain the success of an alternative minimization-like approach.

This result, moreover, provides conditions not only for an arbitrarily small error bound, but also gives an upper bound on the sample complexity needed to achieve the stated guarantee. A key question one might ask is how this bound on sample complexity compares to similar results in the literature and to the information theoretic lower bound. In both cases, the results are positive. In 2012, Jain, et al. exhibited a similar bound with a required sample complexity that was a quartic factor larger than Hardt's result – a major step forward. In addition, this result also holds up well to the information-theoretic lower bound on the sample complexity of $\Omega(k\mu(U)n)$, achieving a bound that is a $O(k||M||_F/\sigma_k)^2$ factor off from the theoretical optimal.

3.6 Demystifying the Algorithm

Up until now, we've highlighted results showing that an alternating minimization-type algorithm achieves provable bounds on its error rate and sample complexity, but we've managed to avoid saying anything about the algorithm itself. We address that fact in this section and a present an intuitive overview of how this algorithm works.

The algorithm, referred to as Smoothed alternating least squares (SAltLS) incorporates several celebrated algorithmic ideas, including alternating minimization, the noisy power method developed by [JNS13], and smoothed analysis. The inputs to the algorithm are the set of observed matrix entries S, a parameter T representing the number of iterations to run the algorithm, an perturbation parameter ε, the target rank k, and the coherence parameter μ. The full algorithm itself is remarkably simple to state as it is a collection of four subroutines.

```
Algorithm: SAltLS
Input: $S, T, k, \varepsilon, \mu$

1. SPLIT $S$ into $T$ equal subsamples
2. INITIALIZE matrix $X_0$
3. For $t = 1, 2, \dots, T$:
    (a) $Y_t \leftarrow$ MEDIANLS($S_t, X_{t-1}, T, k$)
    (b) $X_t \leftarrow$ SMOOTHQR($Y_t, \varepsilon, \mu$)
4. Return matrices $X_T, Y_T$
```

At first glance, it may not seem clear how this algorithm relates to alternating minimization in any way. A closer examination, though, shows that the two steps inside the main loop are indeed our proxy for alternating minimization in the sense that one matrix (between X and Y) is fixed while the other is updated.

The first step of the algorithm, the SPLIT subroutine is designed to partition the collection of observed entries of M so that every new iteration of the alternating step will use fresh, unseen matrix entries to update X and Y. Intuitively, this is a useful property to have for ensuring convergence in just T time steps. Crucially, the SPLIT operation must be defined so that the original probabilistic assumptions from which the elements of S were generated continue to hold after the partition. The details are included in the appendix of the original paper.

Next, we have the INITIALIZE subroutine which simply seeds the matrix X with values at the outset of the algorithm in a manner that encourages fast convergence. Developing good initialization procedures is a major area of work in numerical analysis and, in this case, most canonical initialization procedures work well. In the paper, the authors initialize X by taking a modified truncation of an approximation of the top k singular vectors of M

Now, we discuss the two subroutines in the alternating update steps. The first helper function, MEDIANLS is inspired by the previous work of [JNS13]. on the *noisy power method.* The standard power method, also known as subspace iteration, is a famous algorithm in numerical linear algebra for computing the eigenvectors of a matrix. Typically, the power method is described by first initializing a random vector b_0, and, at each iteration i, multiplying b_i by a matrix A and normalizing by dividing by $||Ab_i||$. In their work, Jain, et al. use the key premise of the power method to replace the least squares alternating minimization objective with a "noisy" power update rule. In particular, the noisy power method consists of the following steps:

Input: An $n \times n$ symmetric matrix M, target rank k and a random orthonormal matrix X_0

1. For $t = 1, 2, \dot{;} T$
   (a) Sample a random noise matrix G_t
   (b) Set $Y_t \leftarrow MX_{t-1} + G_t$
   (c) $X_t =$ ORTHONORMALIZE(Y_t)

Why are we allowed to replace our squared errors update with this single power method calculation? Recall that we added an additional assumption at the outset of presenting the algorithm that the matrix M is square and symmetric. In this case, we can get away without an alternating update since there is no need to "flip" X and Y due to symmetry. Therefore we can rewrite our update as a noisy power method which has very nice convergence properties, formalized in the literature by examining a nation of principal angles between subspaces. Replacing the alternating minimization objective with this noisy power method variant allows us to prove the type convergence guarantees we are aiming for. And, to put the icing on the cake, we don't sacrifice any generality in the process since we can always modify a square matrix into a rectangular one. This sleight of hand is the main trick in both papers for proving exact recovery for matrix completion via alternating minimization and it is quite a clever idea. An additional novelty in Hardt's approach is to compute $O(\log n)$ copies of the matrix Y_t via repeated sampling and then taking the component-wise median of the matrices to ensure a stronger convergence. This is the essence of the MEDIANLS subroutine.

The final subroutine takes its inspiration from the framework of smoothed analysis. The motivation for this smooth orthonormalization step is a desire to maintain the coherence of X_t at each iteration – a fact used in the analysis of the algorithm to achieve the desired recovery result. One natural way of achieving this result would be to show that X_t can be written in terms of Y_t via a QR-decomposition. That is, we hope to write that $X_t = Y_t R^{-1}$ for same invertible, upper triangular matrix R.

The problem is that such a statement is not true in general. However, we can apply a very clever fix using ideas from smoothed analysis of the QR-decomposition. We will add addition Gaussian noise to Y_l which will then allow us to maintain the coherence of X_t. In particular, we will add a matrix H_t to Y_t where the entries of H_t are sampled independently from a Gaussian distribution. The details of verifying that coherence is maintained are non-trivial and we leave them to the original paper. Nevertheless, we hope to emphasize that this component of the algorithm is critical to the exact recovery guarantee and to highlight an ingenious application of smoothed analysis in making a matrix factorization problem tractable.

3.7 Conclusion

We will close this section by reviewing our journey through an algorithm for exact recovery via alternating minimization for the exact matrix complete problem. To recap, we were able to find a set of reasonable conditions under which alternating minimization successfully solves the matrix completion problem by:

- First assuming that the matrix M is symmetric and factors in $U\Lambda U^T$. Recall that this assumption is essentially without loss of generality.
- Replacing the alternating least squares objectives with the noisy power method for which we can prove convergence guarantees
- Introduce the MEDIANLS and SMOOTHEDQR subroutines to achieve an improved recovery guarantee

In conclusion, we have shown that exact recovery via alternating minimization is possible under fairly reasonable assumptions, providing a theoretical explanation for why the overwhelmingly popular heuristic of choice in practice performs well.

4 Alternating Minimization for Dictionary Learning

In this chapter, we will explore the problem of dictionary learning, also known as sparse coding, through the lens of alternating minimization. In continuing our theme of understanding when alternating minimization returns an optimal or near-optimal solution, we will introduce a nifty result due to Agarwal, Anandkumar, Jain, and Netrapalli [AAJ$^+$13] that states a set of conditions under which alternating minimization converges with arbitrarily small error for the dictionary learning problem. Then, we will conclude this chapter by presenting a polynomial time algorithm for certain special cases of the problem due to Arora, Ge, and Moitra [AGMM15].

4.1 Warmup: Sparse Recovery

To motivate the dictionary learning problem, let's review the setup of a closely related problem: sparse recovery. Given a matrix $\mathbf{A}$ with more columns than rows and a vector $\mathbf{b}$, the goal of the sparse recovery problem (at least in this formulation) is to compute the sparsest vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{b}$. This problem is NP-Hard, but – in a similar manner to the results we encountered in the previous chapter – we can impose conditions under which certain algorithms (such as an ℓ_1 norm LP relaxation) can recover the exact solution. This line of thinking will be very useful when we turn our attention to dictionary learning, which, as we shall see, is the mirror image problem to sparse recovery. Before, we proceed, let's introduce a definition of incoherence as it will be easier here to work with incoherence as opposed to the coherence quantity we defined earlier.

Definition 4.1. *Let $A \in \mathbb{R}^{n\times m}$. The columns of A are μ-*incoherent *if for all $i \neq j$*

$$|\langle A_i, A_j\rangle| \leq \mu \|A_i\| \cdot \|A_j\|$$

Where the notation $\langle A_i, A_j\rangle$ denotes the inner product between the column vectors A_i and A_j

4.2 Dictionary Learning

As mentioned previously, dictionary learning can be thought of as, in a sense, the dual problem to sparse recovery [Moi14]. To be precise, suppose we are given many measurements (or training examples) $b_1, b_2, \ldots, b_p \in \mathbb{R}^n$. We would like to find a *dictionary* $A \in \mathbb{R}^{n\times m}$ such that we can write every b_i as a *sparse* linear combination of the columns of A. In the literature, the columns of A are sometimes referred to as the "atoms" of the dictionary, the base vectors from which all linear combinations must be computed

Note the parallels between the problem definition of dictionary learning and sparse recovery. Both insist on sparse solutions, but in dictionary learning we have the power to choose the entries of A which can be learned over time. Formally, we can write the dictionary learning problem as a matrix multiplication of the form

$$Y = A^\star X^\star$$

where

$$Y \in \mathbb{R}^{p \times n} \quad A^\star \in \mathbb{R}^{p \times r} \quad X^\star \in \mathbb{R}^{r \times n}$$

From the previous chapter, we might look at the above matrix multiplication and think that alternating minimization might be useful for this problem as well. This is in fact what most practitioners opt to do to learn the matrix elements. In this case, the alternating minimization algorithm will be of the form:

1. Set initial guess for A
2. Fix A and compute a sparse solution X such that $AX \approx B$ [This is a convex optimization problem]
3. Fix X and compute A that minimizes the Frobenius norm $\|AX - B\|_F$

   As before, the alternating minimization argument will converge to a local optimum because the error only decreases after each iteration.

4.3 Why Alterating Minimization Succeeds for Dictionary Learning

Now that we have formulated the alternating minimization algorithm in the context of dictionary learning, we once again face the theoretical challenge of identifying conditions under which an alternating minimization approach will recover the dictionary A. In this section, we will survey the results of [AAJ$^+$13] who developed precisely such a theory in 2014. We will state the authors' main result, discuss the assumptions and regularity conditions that go into the theorem and finally finish with a quick sketch of the proof.

Theorem 4.1 (Exact Recovery)**.** *Suppose the necessary assumptions and regularity conditions hold. Then the final output of the alternating minimization algorithm $A(t)$ satisfies*

$$\min_{z \in \{-1,1\}} \|zA_i(t) - A_i^\star\|_2 \leq \sqrt{2}\varepsilon_t \qquad \forall i \in \{1, 2, \ldots, r\}$$

The above captures just what we were shooting for – a theorem that guarantees exact recovery under a set of reasonable conditions. The conditions themselves are quite extensive, so we will omit listing them here for brevity, but one can that they reflect very similar conditions we imposed for exact recovery in sparse recovery which further reinforces the connection between the two problems. In particular, we require that the dictionary matrix satisfies the Restricted Isometry Property (RIP) [Can08] along with a minimum number of samples b_i and a sparse coefficient matrix X, among other similar conditions. These assumptions are all very natural and reinforce just why alternating minimization works so well in practice since real-world data typically satisfies these sparsity constraints while an abundance of data ensures that the sample complexity requirements are also met.

It should be noted that the algorithm the authors propose in the paper is not the generic brand of alternating minimization that we have discussed so far. Instead, it is the same iterative algorithm with an additional thresholding function added to enforce the sparsity of

X. In addition, the final algorithm also includes a normalization at the end of each iteration, but these details do very little to alter the fundamental template of the algorithm which is the alternate minimization approach we have discussed in detail.

Proof Sketch
To prove the main result, it suffices to consider just one iteration of the algorithm. Within an iteration, the authors bound the total error from the alternating minimization least squares update and relate this to the error in sparse recovery. This is the only portion of the proof that relies on the Restricted Isometry Property assumption. Then, the authors invoke a stronger error assumption known as the restricted eigenvalue conditions. Under these conditions, one can show with high probability that the error at the update does indeed become arbitrarily small, which completes the argument. Interestingly, the proof does not require any additional machinery behind known error bound conditions in compressed sensing, emphasizing the fact that the hard work in laying out the conditions for exact recovery allowed for a direct application of other well-known results.

4.4 Applications of Dictionary Learning

We conclude this section with some applications of dictionary learning, one of the most important primitives in modern machine learning. In fact, dictionary learning has have heightened attention in recent years with the explosion of interest in deep learning. “Sparse coding layers” are now a common feature in neural network architectures and neural networks can also be trained to solve dictionary learning tasks for downstream tasks. For instance, learning a sparse basis representation can be very valuable in applications such as computer vision and image recognition where denoising can make a significant impact in clarity. In addition, sparse coding has found applications in neuroscience where one often wants to model a brain signal with only a few neurons.

5 Topic Modeling with Provable Guarantees

Topic modeling is a recent area of interest in which, without human intervention, one is able to discern thematic structure for many documents. There are multiple different applications that would benefit from solving this problem efficiently. To name one, news sites often want to organize their content by topic and topic modeling would allow this to occur efficiently.

This problem is **NP**-Hard even for only two topics. As usually follows from **NP**-Hardness, researchers tried tackling this problem with approximation algorithms and this was achieved through SVD, variational inference, and MCMC. We discussed some of these methods in lecture.

In an effort to create provably efficient algorithms, researchers worked on the problem of statistical recovery, which is essentially the idea that data is generated from a perfect model and the problem to solve is to come up with the parameters for the model. [AFH$^+$12] presented an algorithm that assumes that topics are not correlated and [AGM12] presented an algorithm that must meet the separability assumption.

Definition 5.1. *The separability assumption is that every topic must contain at least one anchor word that has non-zero probability in that topic and has zero probability in all other topics.*

The separability assumption is useful because if a document contains an anchor word then one is assured that its topic is from the set of topics that generated the document.

In general, it seems that the separability assumption is more realistic than assuming that there is no correlation between words as is evidenced by [BL07] and [LM06].

We will now consider improvements to the Arora algorithm that are given in [AGH$^+$13] which involves replacing linear programming with a combinatorial anchor selection algorithm, a replacement for matrix inversion with a gradient-based inference model, and an empirical comparison between recovery-based and likelihood-based topic inference. This algorithm with its improvements has the correctness of Gibbs sampling and runs at least an order of magnitude faster.

Let's define V to be the number of words in the vocabulary and K to be the number of topics. Each topic is associated with a multinomial distribution over the words in the vocabulary, which we can denote with A_k.

We can then create a matrix A, with dimensions $V \times K$, with each of the columns being A_k. This matrix represents the word-topic matrix. In the same way, we can create a matrix W, with dimensions $K \times M$, where there are M documents. Thus, W represents the topic-document matrix.

Our task is to determine A (assuming it was created by a perfect data model).

One can use the aforementioned approximation methods, but Arora's algorithm is provably efficient (with separability as a constraint).

We define the variable p to be a value greater than 0 where for each topic k in A, there is some word i such that $A_{i,k} \geq p$ and $A_{i,k'} = 0$ for $k' \neq k$. The word i in this is called an anchor word. This word-topic matrix is therefore p-separable.

As an example, here is a sample matrix A. Note that for the sake of example, there is no underlying distribution in each column, but that can be easily achieved.

Table 1: Matrix A

	food	NBA	health
apple	2	0	1
pilates	0	0	5
basketball	0	4	0
orange	1	0	1
oreos	2	0	0

In this example, we have the topic food having its anchor word be oreos because its value is zero for all other topics and it is greater than 0 for food. Similarly, for NBA, it is basketball and for health, it is pilates.

This example also emphasizes how assuming that correlation between words does not happen is unreasonable. For example, both health and food are highly related because health is often a result of good eating habits.

Using the p from p-separability, D from the length of each of the documents (must be greater than or equal to 2), R which is the $K \times K$ topic-topic covariance matrix, and α_k which is the expected proportion of topic k in a document, we can learn the minimum number of documents to be able to recover a topic model is

$$M = \max\left\{O\left(\frac{\log(V)a^4K^6}{\epsilon^2p^6\gamma^2D}\right), O\left(\frac{\log Ka^2K^4}{\gamma^2}\right)\right\}$$

We don't need to concern ourselves with the details for how this was achieved. But what we should note is that the runtime can be very large since it relies on solving V linear programs. Furthermore, it uses matrix inversion which is noisy because it can end up giving negative values for topic-word probabilities.

This is where the results from this paper come in. There are two steps to the algorithm: anchor selection and recovery. We first create a matrix Q that is of size $V \times V$ that documents word-word co-occurrence counts and then we normalize it.

Original Recovery Problem

Let's take a look at how Arora prevoiusly solved the recovery problem.

First we reorganize Q so that the first K rows and columns correspond to the anchor words. Q_S is the first K rows and $Q_{S,S}$ is the first K rows and first K columns. Then, we run the original recovery algorithm which requires matrix inversion. However, because we run matrix inversions on partial data, namely only the K rows/columns of Q, we end up removing a lot of the data and also relying on co-occurrences between words and their anchors (could be bad if both occur less frequently).

New Recovery Problem

This is where we use a probabilistic approach. Here, we will use $\bar{Q}$, which is the normalized Q. Choose two words w_1 and w_2 and their corresponding topic assignments which we can call z_1 and z_2. Provided that we have infinite data, we know that in our matrix Q we can have $Q_{i,j} = p(w_1 = i, w_2 = j)$. The normalized $\bar{Q}$ matrix can then be $p(w_2 = j|w_1 = i)$ using Bayes Rule.

To clarify, when we have an anchor word s_k, the value of normalized $\bar{Q}_{s_k,j}$ is

$$\sum_w p(z_1 = w|w_1 = s_k)p(w_2 = j|z_1 = w)$$

Because we know that when we have a word w other than k, then $p(z_1 = w|w_1 = s_w) = 0$ (because s_k is an anchor word), this simplifies to

$$\bar{Q}_{s_k,j} = p(w_2 = j|z_1 = k)$$

On the other hand, if s_k is not an anchor word, then we cannot necessarily say this, so we get

$$\bar{Q}_{i,j} = \sum_k^{\prime} p(z_1 = k|w_1 = i)p(w_2 = j|z_1 = k)$$

We know that C is not negative and that its summation is equal to one, so we know that any row of $\bar{Q}$ is in the convex hull of the rows for the anchor words.

Using Bayes' rule, we know that

$$p(w_1 = i|z_1 = k) = \frac{p(z_1 = k|w_1 = i)p(w_1 = i')}{\sum_{i'} p(z_1 = k|w_1 = i')p(w_1 = i')}$$

We also know the values of Q, so we can solve for $p(w_1 = i)$ using the fact that

$$\sum_j Q_{i,j} = \sum p(w_1 = i, w_2 = j) = p(w_1 = i)$$

Using these values, we can then easily find A. The advantage here is that there was no matrix inversion.
The algorithm for recovering A is as follows:

```
Normalize $Q$ to get $\bar{Q}$
Store the normalization constants as $m$
Solve $C_i = \text{ argmin }_{\bar{C}_i} D_{KL}(\bar{Q_i} || \sum_k C_{i,k} \bar{Q_{s_k}})$ for $k$ that are anchor words and that fulfill constraints for $C$
$A' = \text{ diag}(m) \cdot C$
Normalize the columns of $A'$ to get $A$
return $A$
```

Now that we have successfully constructed a more efficient and accurate recover function, let's look at the function to find anchor words.

Earlier, Arora was able to construct an algorithm to find anchor words that involved solving V linear programs. This is the algorithm we covered in lecture. In this paper, however, Arora is able to provide an improvement to this algorithm that uses a combinatorial approach.

This algorithm continually finds the furthest point from the area that is spanned by anchor words found so far. The idea is that when it chooses the furthest points, they will tend to be different from the anchor words chosen so far, which will also help with the guarantee of getting K points that are close to anchor words.

We define P to be the convex hull of V, which is a simplex.

Definition 5.2. *γ-robust means that a simplex P has for each its vertices, the l_2 distance between v and the convex hull of the rest of the vertices is at least γ*

Definition 5.3. *For the set of points whose convex hull is P, then each point ϵ-covers a point in P if it is a convex combination of the vertices.*

Theorem 5.1. *We can find K anchor words in $O(V^2 + \frac{VK}{\epsilon^2})$ that $O(\frac{\epsilon}{\gamma})$ covers the vertices with some arbitrary constraints.*

Definition 5.4. *The span(S) is the space spanned by points in S. The distance from a point to span(S) is the norm of the project of x onto the the orthogonal complement of span(S).*

The algorithm is as follows. In the first phase, we take in the V points and project them. First we start with an empty set S and then we add to it the furthest point from the origin. We then add to S the point that has the largest distance to span(S). We continually do this until we have gone through K points.

The proof for this algorithm uses the fact that even when we run this algorithm for large inputs, we are able to prevent error from accumulating much. The cleanup phase that happens after the previous phase helps with this in removing some error. The cleanup phase runs K times and each time finds a point d_i that has the largest distance to $\text{span}(S/v_i')$. These points d_i are then updated and then returned as anchor words.

```
Project points from V onto subspace
```

```
    $S = d_i$ with $d_i$ being the furthest point from the origin
    add to S points that have the largest distance to span(S) $K - 1$ times
    Find the point that has the largest distance to span(S/$v_i$) and update that entry in $S$. Do
    this $K$ times.
    return $S$
```

Overall, in this paper, Arora et al. were able to show major improvements for solving topic modeling. Now, finding the anchor words and recovering does not require linear programming or rely on V and is provably efficient. This is a major improvement in the field of topic modeling. These advancements in topic modeling are a testament to the fact that even if a problem is **NP**-Hard, with some clever methodologies and assumptions (like separability), one can often recover meaningful results in polynomial time.

References

[AAJ+13] Alekh Agarwal, Animashree Anandkumar, Prateek Jain, Praneeth Netrapalli, and Rashish Tandon. Learning sparsely used overcomplete dictionaries via alternating minimization. *CoRR*, abs/1310.7991, 2013.

[AFH+12] Animashree Anandkumar, Dean P. Foster, Daniel J. Hsu, Sham M. Kakade, and Yi-Kai Liu. Two svds suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. *CoRR*, abs/1204.6703, 2012.

[AGH+13] Sanjeev Arora, Rong Ge, Yonatan Halpern, David M. Mimno, Ankur Moitra, David Sontag, Yichen Wu, and Michael Zhu. A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 280–288, 2013.

[AGM12] Sanjeev Arora, Rong Ge, and Ankur Moitra. Learning topic models - going beyond SVD. *CoRR*, abs/1204.1956, 2012.

[AGMM15] Sanjeev Arora, Rong Ge, Tengyu Ma, and Ankur Moitra. Simple, efficient, and neural algorithms for sparse coding. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 113–149, 2015.

[BE02] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[BL07] David M Blei and John D Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35, 2007.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[Can08] Emmanuel J Candes. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9-10):589–592, 2008.

[DHS11] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[FGKS15] Roy Frostig, Rong Ge, Sham M. Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 728–763, 2015.

[Har13] Moritz Hardt. On the provable convergence of alternating minimization for matrix completion. *CoRR*, abs/1312.0925, 2013.

[HK14] Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *Journal of Machine Learning Research*, 15(1):2489–2512, 2014.

[HKKA06] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. In *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, pages 499–513, 2006.

[HRS15] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240, 2015.

[JNS13] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 665–674, 2013.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KH91] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 950–957, 1991.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[LM06] Wei Li and Andrew McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 577–584, 2006.

[MM16] Konstantin Makarychev and Yury Makarychev. Metric perturbation resilience. *CoRR*, abs/1607.06442, 2016.

[MMV14] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu-linial stable instances of max cut and minimum multiway cut. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 890–906, 2014.

[Moi14] Ankur Moitra. Algorithmic aspects of machine learning. *Lecture notes*, 2014.

[Nes13] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer Science & Business Media, 2013.

[NJLS09] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[NY78] Arkadi Nemirovski and D Yudin. On cezari's convergence of the steepest descent method for approximating saddle point of convex-concave functions. In *Soviet Math. Dokl*, volume 19, 1978.

[NYD83] Arkadi Nemirovskii, David Borisovich Yudin, and Edgar Ronald Dawson. Problem complexity and method efficiency in optimization. 1983.

[RRWN11] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 693–701, 2011.

[SHK$^+$14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[SZOR15] Christopher De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2674–2682, 2015.