Minimum weight norm models do not always generalize well for over-parameterized problems

Vatsal Shah  
University of Texas at Austin  
vatsalshah1106@utexas.edu

Anastasios Kyrillidis  
Rice University  
anastasios@rice.edu

Sujay Sanghavi  
University of Texas at Austin  
sanghavi@mail.utexas.edu

February 13, 2019

Abstract

Stochastic gradient descent is the de facto algorithm for training deep neural networks (DNNs). Despite its popularity, it still requires fine tuning in order to achieve its best performance. This has led to the development of adaptive methods, that claim automatic hyper-parameter optimization. Recently, researchers have studied both algorithmic classes via toy examples: e.g., for over-parameterized linear regression, Wilson et al. (2017) shows that, while SGD always converges to the minimum-norm solution, adaptive methods show no such inclination, leading to worse generalization capabilities.

Our aim is to study this conjecture further. We empirically show that the minimum weight norm is not necessarily the proper gauge of good generalization in simplified scenarios, and different models found by adaptive methods could outperform plain gradient methods. In practical DNN settings, we observe that adaptive methods can outperform SGD, with larger weight norm output models, but without necessarily reducing the amount of tuning required.

1 Introduction

In theory, deep neural networks (DNNs) are hard to train Blum & Rivest (1989). Apart from sundry architecture configurations – such as network depth, layer width, type of activation – there are key algorithmic hyper-parameters that need to be properly tuned, in order to obtain a model that generalizes well, within reasonable amount of time.

Among the hyper-parameters, the one of pivotal importance is the step size Sutskever et al. (2013) Schaul et al. (2013). To set the background, most algorithms in practice are gradient-descent based: given the current model estimate $w_k$ and some training examples, we iteratively compute the gradient $\nabla f(w_k)$ of the objective $f(\cdot)$, and update the model by advancing along negative directions of the gradient $-\nabla f(w_k)$, weighted by the step size $\eta$; i.e.,

$$w_{k+1} = w_k - \eta \nabla f(w_k).$$

This is the crux of all gradient-descent based algorithms, including the ubiquitous stochastic gradient descent (SGD) algorithm. Here, the step size could be set as constant, or could be changing per iteration $\eta_k$ Bottou (2012), usually based on a predefined learning rate schedule Bottou (2010), Xu (2011), Senior et al. (2013). Beyond practical strategies and tricks that lead to faster convergence and better generalization Bengio (2012), Orr & Müller (2003), Bottou (2012), during the past decade we have witnessed a family of algorithms that argue for automatic hyper-parameter adaptation Ruder (2016) during training (including step size). The list includes AdaGrad Duchi et al. (2011), Adam Kingma & Ba (2014), AdaDelta Zeiler (2012), RMSProp
Tieleman & Hinton (2012), AdaMax Kingma & Ba (2014), Nadam Dozat (2016), just to name a few. These algorithms utilize current and past gradient information \(\{\nabla f(w_t)\}_{t=1}^k\), for \(t < k\), to design preconditioning matrices \(D_k \succeq 0\) that better pinpoint the local curvature of the objective function, as follows:

\[
w_{k+1} = w_k - \eta D_k \nabla f(w_k).
\]

The main argument is that \(D_k\) eliminates pre-setting a learning rate schedule, or diminishes initial bad step size choices, thus, detaching the time-consuming part of step size tuning from the practitioner Zhang & Mitliagkas (2017).

Recently though, it has been argued that simple gradient-based algorithms may perform better compared to adaptive ones. More specifically, for the linear regression setting, Wilson et al. (2017) shows that, under specific assumptions, the adaptive methods converge to a different solution than the minimum norm one. The latter has received attention due to its efficiency as the maximum margin solution in classification Poggio et al. (2017). This behavior is also demonstrated using DNNs, where i) simple gradient descent generalizes at least as well as the adaptive methods, ii) adaptive techniques require at least the same amount of tuning as the simple gradient descent methods.

In this paper, we further study this conjecture. The paper is separated into the theoretical (Sections 2-4) and the practical (Subsection 4.3 and Section 5) part. For our theory, we focus on simple linear regression (Section 2), and discuss the differences between under- and over-parameterization. Section 3 focuses on simple gradient descent, and establishes closed-form solutions, for both settings. We study the AdaGrad algorithm on the same setting in Section 4, and we discuss under which conditions gradient descent and AdaGrad perform similarly or their behavior diverges.

Our findings can be summarized as follows:

- For under-parameterized linear regression, closed-form solutions indicate that simple gradient descent and AdaGrad converge to the same solution; thus, adaptive methods have the same generalization capabilities, which is expected in convex linear regression.

- In over-parameterized linear regression, simple gradient methods converge to the minimum norm solution. In contrast, AdaGrad converges to a different point; while the computed model fits the training data, it has different generalization behavior on unseen data, than the minimum norm solution. Wilson et al. (2017) shows that AdaGrad generalizes worse than gradient descent; in this work, we empirically show that an AdaGrad variant generalizes better than the minimum norm solution on a different counterexample. We conjecture that the superiority of simple or adaptive methods depends on the problem/data at hand, and the discussion “who is provably better” is inconclusive.

- We conduct neural network experiments using different datasets and network architectures. Overall, we observe a similar behavior either using simple or adaptive methods. Our findings support the conclusions of Wilson et al. (2017) that adaptive methods still require fine parameter tuning. Generalization-wise, we observe that simple algorithms are not universally superior than adaptive ones.

## 2 Background on linear regression

Consider the linear regression setting:

\[
\min_{w \in \mathbb{R}^d} \frac{1}{2} \cdot \|X w - y\|_2^2,
\]

where \(X \in \mathbb{R}^{n \times d}\) is the feature matrix and \(y \in \mathbb{R}^n\) are the observations. There are two different settings, depending on the number of samples and dimensions:

- **Over-parameterized case**: We have more parameters than the number of samples: \(d \geq n\). In this case, assuming that \(X\) is in general position, \(XX^\top\) is full rank.

- **Under-parameterized case**: Here, the number of samples is larger than the number of parameters: \(n \geq d\). In this case, usually \(X^\top X\) is full rank.

\[1\]In this work, our theory focuses on adaptive but non-momentum-based methods, such as AdaGrad.
The most studied case is when \( n \geq d \): the problem has solution \( w^* = (X^T X)^{-1}X^T y \), under full rankness assumption on \( X \). In the case where the problem is over-parameterized \( d \geq n \), there is a solution of similar form that has received significant attention, despite the infinite number of optimal solutions: This is the so-called minimum norm solution. The optimization instance to obtain this solution is:

\[
\min_{w \in \mathbb{R}^d} \|w\|_2^2 \quad \text{subject to} \quad y = Xw.
\]

Let us denote its solution as \( w_{mn} \), which has the form \( w_{mn} = X^T (XX^T)^{-1} y \). Any other solution has to have equal or larger Euclidean norm than \( w_{mn} \).

Observe that the two solutions, \( w^* \) and \( w_{mn} \), differ between the two cases: in the under-parameterized case, the matrix \( X^T X \) is well-defined (full-rank) and has an inverse, while in the over-parameterized case, the matrix \( XX^T \) is full rank. Importantly, there are differences on how we obtain these solutions in an iterative fashion. We next show how both simple and adaptive gradient descent algorithms find \( w^* \) for well-determined systems. This does not hold for the over-parameterized case: there are infinite solutions, and the question which one they select is central in the recent literature \cite{Wilson2017, Nacson2018, Gunasekar2018}.

### 3 Closed-form expressions for gradient descent in linear regression

Studying iterative routines in simple tasks provides intuitions on how they might perform in more complex problems, such as neural networks. Next, we set the background with the well-known under-parameterized linear regression, before we move onto the over-parameterized case.

#### 3.1 Under-parameterized linear regression.

Here, \( n \geq d \) and \( X^T X \) is assumed to be full rank. Simple gradient descent with step size \( \eta > 0 \) satisfies:

\[
w_{k+1} = w_k - \eta \cdot \nabla f(w_k) = w_k - \eta X^T (Xw_k - y).
\]

Unfolding for \( K \) iterations, we get (Section A.1):

\[
w_K = \left( \sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (X^T X)^{i-1} \right) X^T y.
\]

The expression in the parentheses satisfies:

\[
\sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (X^T X)^{i-1} = (-X^T X)^{-1} \cdot ((I - \eta X^T X)^K - I)
\]

Therefore, we get the closed form solution:

\[
w_K = (-X^T X)^{-1} \cdot ((I - \eta X^T X)^K - I) \cdot X^T y.
\]

In order to prove that gradient descent converges to the minimum norm solution, we need to prove that:

\[
(I - \eta X^T X)^K - I \rightarrow -I \Rightarrow (I - \eta X^T X)^K \rightarrow n,K \rightarrow 0.
\]

This is equivalent to showing that \( \| (I - \eta X^T X)^K \|_2 \rightarrow 0 \). From optimization theory \cite{Nesterov2013}, we need \( \eta < \frac{1}{\lambda_i(X^T X)} \) for convergence, where \( \lambda_i(\cdot) \) denotes the eigenvalues of the argument. Then, \( H := I - \eta X^T X \in \mathbb{R}^{d \times d} \) has spectral norm that is smaller than 1, i.e., \( \| H \| \leq 1 \). Combining the above, we use the following theorem.

**Theorem 1** \cite{Horn1985, Dowler2013} \textbf{Behavior of square matrix} \( \| M^K \|_2 \rightarrow \| H \| \rightarrow 0 \). Let \( M \) be a \( d \times d \) matrix. Let \( \tau(M) = \max_i |\lambda_i(M)| \) denote the spectral radius of the matrix \( M \). Then, there exists a sequence \( \varepsilon_K \geq 0 \) such that: \( \| M^K \|_2 \leq (\tau(M) + \varepsilon_K)^K \), and \( \lim_{K \rightarrow \infty} \varepsilon_K = 0 \).

Using the above theorem, \( H \) has \( \tau(H) < 1 \). Further, for sufficiently large \( k < K \), \( \varepsilon_K \) has a small value such that \( \tau(H) + \varepsilon_K < 1 \); i.e., after some \( k_1 < K \), \( (\tau(H) + \varepsilon_{k_1})^{k_1} \) will be less than zero, converging to zero for increasing \( k_1 \). As \( K \) is going towards infinity, this concludes the proof, and leads to the left inverse solution:

\[
w_\infty = (-X^T X)^{-1} \cdot (-I) X^T y = (X^T X)^{-1} X^T y \equiv w^* \text{, as } K \rightarrow \infty.
\]

This is identical to the closed form solution of linear regression.
3.2 Over-parameterized linear regression.

For completeness, we briefly provide the analysis for the over-parameterized setting, where $d \geq n$ and $XX^\top$ is assumed to be full rank. By inspection, unfolding gradient descent recursion gives:

$$w_K = X^\top \left( \sum_{i=1}^{K} (-1)^{i-1} \cdot \left( \begin{array}{c} K \\ i \end{array} \right) \cdot \eta^i \cdot (XX^\top)^{i-1} \right) y.$$  

Similarly, the summation can be simplified to:

$$\sum_{i=1}^{K} (-1)^{i-1} \cdot \left( \begin{array}{c} K \\ i \end{array} \right) \cdot \eta^i \cdot (XX^\top)^{i-1} = (-XX^\top)^{-1} \cdot ((I - \eta XX^\top)^K - I),$$

and, therefore:

$$w_K = X^\top (-XX^\top)^{-1} \cdot ((I - \eta XX^\top)^K - I) y.$$  

Under similar assumption on the spectral norm of $(I - \eta XX^\top)^K$ and using Theorem 1, we obtain the right inverse solution: $w_\infty = X^\top(-XX^\top)^{-1} \cdot (-I) y = X^\top(XX^\top)^{-1} y \equiv w_{\text{min}}$, as $K \to \infty$.

Bottomline, in both cases, gradient descent converges to left and right inverse solutions.

4 Closed-form expressions for adaptive gradient descent in linear regression

For simplicity, we study non-accelerated adaptive gradient descent methods, like AdaGrad, following the analysis in Wilson et al. (2017); the momentum-based schemes are left for future work, but this is sufficient for our goal to prove that preconditioned methods perform as well as plain methods. While there exists considerable work analyzing the stochastic variants of adaptive methods in Duchi et al. (2011), Kingma & Ba (2014), Ward et al. (2018), Mukkamala & Hein (2017), we concentrate on non-stochastic variants, for simplicity and ease of comparison with gradient descent. In summary, we study: $w_{k+1} = w_k - \eta D_k \cdot \nabla f(w_k), \quad D_k > 0, \forall k$. E.g., in the case of AdaGrad, we have:

$$D_k = \text{diag}\left( 1/ \sqrt{ \sum_{j=k-J}^{k} \nabla f(w_j) \odot \nabla f(w_j) + \varepsilon} \right) > 0,$$

for some $\varepsilon > 0$, and $J < k \in \mathbb{N}_+$. The main ideas apply for any positive definite preconditioner. The case where $D_k = D$, for $D$ a constant matrix, is deferred to the appendix (Section A.2).

4.1 Under-parameterized linear regression.

When $D_k$ is varying (Section A.3), we end up with the following proposition (folklore); see Section A.4:

**Proposition 1** Consider the under-parameterized case. Assume the recursion $w_{k+1} = w_k - \eta D_k \nabla f(w_k)$, for $D_k > 0$. Then, after $K$ iterations, $w_K$ satisfies:

$$w_K = (-X^\top X)^{-1} \left( \prod_{i=K-1}^{0} (I - \eta X^\top XD_i) - I \right) X^\top y.$$  

Using Theorem 1 we can infer that, for sufficiently large $K$ and for sufficiently small $\eta < \max_i \frac{1}{\lambda_i(X^\top XD_i)}$, such that $\|I - \eta X^\top XD_i\| < 1 \forall i$, we have: $\prod_{i=K-1}^{0} (I - \eta X^\top XD_i) \to 0$. Thus, for sufficiently large $K$ and assuming $\eta < \max_i \frac{1}{\lambda_i(X^\top XD_i)}, \forall i$: $w_\infty = (X^\top X)^{-1} \cdot X^\top y \equiv w^*$, which is the same as the plain gradient descent approach. Thus, in this case, under proper $\eta$ assumptions (which might seem stricter than plain gradient descent), adaptive methods have the same generalization capabilities as gradient descent.
4.2 Over-parameterized linear regression.

Let us focus on the case where $n < d$. Finding a closed form for $w_K$ seems much trickier to achieve. Here, we follow a different path than the previous sections.

What is the predictive power of adaptive methods within the training set?

For the first question, we look for a way to express the predictions within the training dataset, i.e., $\hat{y}_K = Xw_K$, where $w_K$ is found by the recursion $w_{k+1} = w_k - \eta D_k \nabla f(w_k)$.

**Proposition 2** Consider the over-parameterized case. Assume the recursion $w_{k+1} = w_k - \eta D_k \nabla f(w_k)$, for $D_k > 0$. Then, after $K$ iterations, the prediction $\hat{y}_K$ satisfies:

$$\hat{y}_K = Xw_K = -\left( \prod_{i=K-1}^{0} (I - \eta XD_iX^\top) - I \right)y.$$

The proof can be found in Section A.5. Using Theorem 1, we observe that, for sufficiently large $K$ and for sufficiently small step size $\eta < \max \frac{1}{\lambda_{\min}(XDX^\top)}$, $\|I - \eta XD_iX^\top\| < 1 \forall i$. Thus, $\prod_{i=K-1}^{0} (I - \eta XD_iX^\top) \to 0$. This further implies that $\hat{y}_K = y$, as $K$ increases; i.e., adaptive methods fit the training data, and make the correct predictions within the training dataset.

What is the predictive power of adaptive methods on unseen data?

We start with the counterexample in Wilson et al. (2017), where adaptive methods – where $D_k$ takes the form of (1) – fail to find a solution that generalizes, in contrast to gradient descent methods (under assumptions).

Let us briefly describe their setting: we take $d$ to be of the order of $cn$, with $c > 1$; empirically, the counterexample holds for various values of $n$, as long as $d > n$. For the responses, we consider two classes $y_i \in \{\pm 1\}$. For $i = 1, \ldots, n$, we sample $y_i$ with probability $p$ as $+1$, and with probability $1 - p$ as $-1$, for $p > 1/2$. Given $y_i$, for each $i$, we design the $i$-th row of $X$, $X_i$ as:

$$(X_i)_j = \begin{cases} y_i, & j = 1, \\ 1, & j = 2, 3, \\ 1, & j = 4 + 5(i-1), \\ 0, & \text{otherwise}, \end{cases} \quad \text{if } y_i = 1,$$

$$(X_i)_j = \begin{cases} y_i, & j = 1, \\ 1, & j = 2, 3, \\ 1, & j = 4 + 5(i-1), \\ \cdots, & 8 + 5(i-1), \\ 0, & \text{otherwise}. \end{cases} \quad \text{if } y_i = -1.$$

Given this structure for $X$, only the first feature is indicative for the predicted class: i.e., one model that always predicts correctly is $w^* = (1, 0, 0, \ldots, 0) \in \mathbb{R}^d$; however, we note that this is not the only model that might lead to the correct predictions. The 2nd and 3rd features are the same $\forall X_i$, and the rest features are unique for each $X_i$.

Given this generative model and assuming $p > 1/2$, Wilson et al. (2017) show that AdaGrad, with $D_k$ as in (1), only predicts correctly the positive class, while plain gradient descent-based schemes perform flawlessly (predicting both positive and negative classes correctly), as long as the number of positive examples in training is $> 1/3$ of the negative ones. This shows that simple gradient descent generalizes better than adaptive methods for this simple problem instance; this further implies that such behavior might transfer to more complex cases, such as neural networks.
Table 1: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and adaptive gradient descent methods. We set $p = 7/8$ and $J = 10$, as in the main text. The adaptive method uses $D_k$ according to (2). The distances shown are median values out of 100 different realizations for each setting; the accuracies are obtained by testing $10^4$ predictions on unseen data.

| $n$ | $\ell = 1/32$ | $\ell = 1/16$ | $\ell = 1/8$ |
|-----|---------------|---------------|---------------|
|     | $\frac{\|\hat{w} - w_{mn}\|_2}{\|w_{mn}\|_2}$ | $\frac{\|\hat{w} - w_{mn}\|_2}{\|w_{mn}\|_2}$ | $\frac{\|\hat{w} - w_{mn}\|_2}{\|w_{mn}\|_2}$ |
| 10  | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ |
|     | $1.015 \cdot 10^{-16}$ | $1.7401 \cdot 10^{-16}$ | $4.08 \cdot 10^{-16}$ |
|     | $63$ | $53$ | $58$ |
| 50  | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ |
|     | $4.729 \cdot 10^{-15}$ | $6.9197 \cdot 10^{-15}$ | $9.1710 \cdot 10^{-15}$ |
|     | $77$ | $80$ | $91$ |
| 100 | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ | $\frac{\text{Acc.} (\%)}{100}$ |
|     | $4.975 \cdot 10^{-9}$ | $2.5420 \cdot 10^{-9}$ | $1.5572 \cdot 10^{-11}$ |
|     | $85$ | $83$ | $100$ |

4.3 A counterexample for the counterexample

To prove otherwise, we could either find a completely different counterexample (since adaptive methods do not perform well in the one in Wilson et al. (2017)), or we find an alternative adaptive method. We follow the second path. We alter the previous counterexample only slightly: we reduce the margin between the two classes; the case where we increase the margin is provided in the appendix. We empirically show that gradient-descent methods fail to generalize as well as adaptive methods—with variant of AdaGrad.

In particular, we consider two classes $y_i \in \{\pm \ell\}$ for some $\ell \in (0, 1)$; i.e., we consider a smaller margin between the two classes. $\ell$ can take different values, and still we get the same performance, as we show in the experiments below. The rest of the problem setting is the same. Likewise as above, only the first feature is indicative of the correct class.

Given this generative model, we construct $n$ samples $\{y_i, x_i\}_{i=1}^n$, and set $d = 6n$, for different $n$ values. We compare two simple algorithms: i) the plain gradient descent for $\eta = \ell/\lambda_i(x^\top x)$; ii) the recursion $u_{k+1} = u_k - \eta D_k X^\top (X u_k - y)$, where $\eta$ is as above, and $D_k$:

$$D_k = \text{diag} \left( \frac{1}{(\sum_{j=k-J}^{k} \nabla f(w_j) \circ \nabla f(w_j) + \varepsilon)^2} \right) > 0,$$

for some $\varepsilon > 0$, and $J < k \in \mathbb{N}^+$.

Observe that $D_k$ uses the dot product of gradients, squared. A variant of this preconditioner is found in Mukkamala & Hein (2017); however our purpose is not to recommend a particular preconditioner but to show that there are $D_k$ that lead to better performance than the minimum norm solution. We denote as $w_{\text{ada}}, w_{\text{adam}}$ and $w_{\text{GD}}$ the estimates of the adam, adagrad variant and simple gradient descent, respectively.

The experiment obeys the following steps: i) we train both gradient and adaptive gradient methods on the same training set; ii) we test models on new data $\{y_i^{\text{test}}, x_i^{\text{test}}\}_{i=1}^Q$. We define performance in terms of the classification error: for a new sample $\{y_i^{\text{test}}, x_i^{\text{test}}\}$ and given $w_{\text{ada}}, w_{\text{adam}}$ and $w_{\text{GD}}$, the only features that

One can consider classes in $\{\pm 1\}$, but the rest of the problem settings need to be weighted accordingly. We selected to weight the classes differently in order not to drift much from the counterexample from Wilson et al. (2017).
are non-zeros in both \( x_i^{\text{test}} \) and \( w \)'s are the first 3 entries (pp. 5, Wilson et al. 2017). This is due to the fact that, for gradient descent and given the structure in \( X \), only these 3 features affects the performance of gradient descent. Thus, the decision rules for both algorithms are:

\[
\hat{y}_i^{\text{adam}} = \text{quant}_\ell \left( w_1^{\text{adam}} \cdot y_i^{\text{test}} + w_2^{\text{adam}} + w_3^{\text{adam}} \right), \\
\hat{y}_i = \text{quant}_\ell \left( w_1 \cdot y_i^{\text{test}} + w_2 + w_3 \right), \\
\hat{y}_i^{\text{GD}} = \text{quant}_\ell \left( w_1^{\text{GD}} \cdot y_i^{\text{test}} + w_2^{\text{GD}} + w_3^{\text{GD}} \right),
\]

where \( \text{quant}_\ell(\alpha) \) finds the nearest point w.r.t. \( \{\pm \ell\} \).

Table 1 summarises the empirical findings. In order to cover a wide range of settings, we set \( n = [10, 50, 100] \) and \( d = 6n \), as dictated by Wilson et al. 2017. We generate \( X \) as above, where instances in the positive class, \( y_i \in +\ell \), are generated with probability \( p = 7/8 \); the cases where \( p = 5/8 \) and \( p = 3/8 \) are provided in the appendix Section A.7, and also convey the same message as in Table 1.

The simulation is completed as follows: For each setting \((n, p, J)\), we generate 100 different instances for \((X, y)\), and for each instance we compute the solutions from gradient descent, AdaGrad variant and Adam (RMSprop is included in the Appendix) and the minimum norm solution \( w_{\text{min}} \). In the appendix, we have the above table with the Adagrad variant that normalizes the final solution \( \hat{w} \) (Table 3) before calculating the distance w.r.t. the minimum norm solution: we observed that this step did not improve or worsen the performance, compared to the unnormalized solution. This further indicates that there is an infinite collection of solutions –with different magnitudes– that lead to better performance than plain gradient descent; thus our findings are not a pathological example where adaptive methods work better.

We record \( \|\hat{w} - w_{\text{min}}\|_2 \), where \( \hat{w} \) represents the corresponding solutions obtained by the algorithms in the comparison list. For each \((X, y)\) instance, we further generate \( \{y_1^{\text{test}}, x_1^{\text{test}}\}_{i=1}^{100} \), and we evaluate the performance of both models on predicting \( y_i^{\text{test}}, \forall i \).

Table 1 shows that gradient descent converges to the minimum norm solution, in contrast to the adaptive methods. This justifies the fact that the adaptive gradient methods (including the proposed adagrad variant) converge to a different solution than the minimum norm solution. Nevertheless, the accuracy on unseen data is higher in the adaptive methods (both our proposed AdaGrad variant and in most instances, Adam), than the plain gradient descent, when \( \ell \) is small: the adaptive method successfully identifies the correct class, while gradient descent only predicts one class (the positive class; this is justified by the fact that the accuracy obtained is approximately close to \( p \), as \( n \) increases).

The proposed AdaGrad variant described in equation 2 falls under the broad class of adaptive algorithms with \( D_k \). However, for the counter example in (pp. 5, Wilson et al. 2017), the AdaGrad variant neither satisfies the convergence guarantees of Lemma 3.1 there, nor does it converge to the minimum norm solution evidenced by its norm in Table 1. To buttress our claim that the AdaGrad variant in (2) converges to a solution different than that of minimum norm (which is the case for plain gradient descent), we provide the following proposition for a specific class of problem:

**Proposition 3** Suppose \( X^\top y \) has no zero components. Define \( Q = \text{diag}(|X^\top y|^3) \) and assume there exists a scalar \( c \) such that \( XQ^{-1}\text{sign}(X^\top y) = cy \). Then, when initialized at \( 0 \), the AdaGrad variant in (2) converges to the unique solution \( w \propto Q^{-1}\text{sign}(X^\top y) \).

This result, combined with our experiments, indicate that the minimum norm solution does not guarantee better generalization performance for over-parameterized settings, even in cases of linear regression. Thus, it is unclear why that should be the case for deep neural networks. A detailed analysis about the class of counter-examples is available in Section A.7.1.

## 5 Experiments

We empirically compare two classes of algorithms:

- Plain gradient descent algorithms, including the mini-batch stochastic gradient descent and the accelerated stochastic gradient descent, with constant momentum.

3Further experiments were performed in Section A.7 to empirically verify this consistency.
4Not the problem proposed in the counter-example 1 on pg 5.
Table 2: Summary of the datasets and the architectures used for experiments. CNN stands for convolutional neural network, FF stands for feed forward network. More details are given in the main text.

| Name  | Network type          | Dataset     |
|-------|-----------------------|-------------|
| M1-UP | Shallow CNN + FFN     | MNIST       |
| M1-OP | Shallow CNN + FFN     | MNIST       |
| C1-UP | Shallow CNN + FFN     | CIFAR-10    |
| C1-OP | ResNet18              | CIFAR-10    |
| C2-OP | PreActResNet18        | CIFAR-100   |
| C3-OP | MobileNet             | CIFAR-100   |
| C4-OP | MobileNetV2           | CIFAR-100   |
| C5-OP | GoogleNet             | CIFAR-100   |

Adaptive methods like AdaGrad [Duchi et al. (2011)], RMSProp [Tieleman & Hinton (2012)], and Adam [Kingma & Ba (2014)], and the AdaGrad variant. Our purpose is not to promote the variant as a better method; it is included for completeness.

The details of the datasets and the DNN architectures used in our experiments are given in Table 2.

5.1 Hyperparameter tuning

Both for adaptive and non-adaptive methods, the step size and momentum parameters are key for favorable performance, as also concluded in [Wilson et al. (2017)]. Default values were used for the remaining parameters. The step size was tuned over an exponentially-spaced set \{0.0001, 0.001, 0.01, 0.1, 1\}, while the momentum parameter was tuned over the values of \{0, 0.1, 0.25, 0.5, 0.75, 0.9\}. We observed that step sizes and momentum values smaller/bigger than these sets gave worse results. Yet, we note that a better step size could be found between the values of the exponentially-spaced set. The decay models were similar to the ones used in [Wilson et al. (2017)]: no decay and fixed decay. We used fixed decay in the over-parameterized cases, using the StepLR implementation in pytorch. We experimented with both the decay rate and the decay step in order to ensure fair comparisons with results in [Wilson et al. (2017)]. A complete set of hyperparameters tuned over for comparison can be found in Section A.8 in the Appendix.

5.2 Results

Our main observation is that, both in under- or over-parameterized cases, adaptive and non-adaptive methods converge to solutions with similar testing accuracy: the superiority of simple or adaptive methods depends on the problem/data at hand. Further, as already pointed in [Wilson et al. (2017)], adaptive methods often require similar parameter tuning. Most of the experiments involve using readily available code from GitHub repositories. Since increasing/decreasing batch-size affects the convergence [Smith et al. (2017)], all the experiments were simulated on identical batch-sizes. Finally, our goal is to show performance results in the purest algorithmic setups: often, our tests did not achieve state of the art performance.

Overall, despite not necessarily converging to the same solution as gradient descent, adaptive methods generalize as well as their non-adaptive counterparts. In M1 and C1-UP settings, we compute standard deviations from all Monte Carlo instances, and plot them with the learning curves (shown in shaded colors is the one-apart standard deviation plots; best illustrated in electronic form). For the cases of C\{1-5\}-OP, we also show the weight norms of the solutions (as in Euclidean distance $\| \cdot \|_2$ of all the trainable weights in the network). Such measure has been in used in practice [Bansal et al. (2018)], as a regularization to find minimum Euclidean norm solutions, inspired by the results from support vector machines [Belkin et al. (2018)].

MNIST dataset and the M1 architecture. Each experiment for M1 is simulated over 50 epochs and 10 runs for both under- and over-parameterized settings. Both the MNIST architectures consisted of two convolutional layers (the second one with dropouts [Srivastava et al. (2014)]) followed by two fully connected layers. The primary difference between the M1-OP ($\sim$ 73K parameters) and M1-UP ($\sim$ 21K parameters)
Figure 1: Accuracy results on unseen data, for different NN architectures and datasets. Top row: Under-parameterized problems; Bottom row: Over-parameterized problems. Left two panels: Accuracy and training loss for MNIST; Right two panels: Accuracy and training loss for CIFAR10.

architectures was the number of channels in the convolutional networks and # of nodes in the last fully connected hidden layer.

Figure 1 left two columns, reports the results over 10 Monte-Carlo realizations. Top row corresponds to the M1-UP case; bottom row to the M1-OP case. We plot both training errors and the accuracy results on unseen data. For the M1-UP case, despite the grid search, observe that AdaGrad (and its variant) do not perform as well as the rest of the algorithms. Nevertheless, adaptive methods (such as Adam and RMSProp) perform similarly to simple SGD variants, supporting our conjecture that each algorithm requires a different configuration, but still can converge to a good local point; also that adaptive methods require the same (if not more) tuning. For the M1-OP case, SGD momentum performs less favorably compared to plain SGD, and we conjecture that this is due to non-optimal tuning. In this case, all adaptive methods perform similarly to SGD.

CIFAR10 dataset and the C1 architecture. For C1, C1-UP is trained over 350 epochs, while C1-OP was trained over 200 epochs. The under-parameterized setting is on-purpose tweaked to ensure that we have fewer parameters than examples (∼43K parameters), and slightly deviates from McDonnell & Vladusich (2015); our generalization guarantees (∼76%) are in conjunction with the attained test accuracy levels. Similarly, for the C1-OP case, we implement a Resnet [He et al., 2016a] + dropout architecture (∼0.25 million parameters) and obtained top-1 accuracy of ∼93%. Adam and RMSProp achieves the best performance than their non-adaptive counterparts for both the under-parameterized and over-parameterized settings.

Figure 1 right panel, follows the same pattern with the MNIST data; it reports the results over 10 Monte-Carlo realizations. Again, we observe that AdaGrad methods do not perform as well as the rest of the algorithms. Nevertheless, adaptive methods (such as Adam and RMSProp) perform similarly to simple SGD variants.

CIFAR100 and other deep architectures (C{2-5}-OP). In this experiment, we focus only on the over-parameterized case: DNNs are usually designed over-parameterized in practice, with ever growing number of layers, and, eventually, a larger number of parameters Telgarsky (2016). We again completed 10 runs for each of the set up we considered. C2-OP corresponds to PreActResNet18 from [He et al., 2016b], C3-OP corresponds to MobileNet from [Howard et al., 2017], C4-OP is MobileNetV2 from [Sandler et al., 2018], and C5-OP is GoogleNet from [Szegedy et al., 2015]. The results are depicted in Figure 2. After a similar hyper-parameter tuning phase, we selected the best choices among the parameters tested. The results show no clear winner once again, which overall support our claims: the superiority depends on the problem/data at hand; also, all algorithms require fine tuning to achieve their best performance. We note that

[5]The code from the following github repository was used for experiments: https://github.com/kuangliu/pytorch-cifar
Figure 2: Accuracy results on unseen data, for different NN architectures on CIFAR100. **Left panel:** Accuracy and weight norm $\|\cdot\|_2$ of trainable parameters for PreActResNet18 in [He et al. (2016b)]. **Left middle panel:** Accuracy and weight norm $\|\cdot\|_2$ of trainable parameters for MobileNet in [Howard et al. (2017)]. **Right middle panel:** Accuracy and weight norm $\|\cdot\|_2$ of trainable parameters for MobileNetV2 in [Sandler et al. (2018)]. **Right panel:** Accuracy and weight norm $\|\cdot\|_2$ of trainable parameters for GoogleNet in [Sandler et al. (2018)].

a more comprehensive reasoning requires multiple runs for each network, as other hyper-parameters (such as initialization) might play significant role in closing the gap between different algorithms.

An important observation of Figure 2 comes from the bottom row of the panel. There, we plot the Euclidean norm $\|\cdot\|_2$ of all the trainable parameters of the corresponding neural network. While such a norm could be considered arbitrary (e.g., someone could argue other types of norms to make more sense, like the spectral norm of layer), we use the Euclidean norm as $i$ it follows the narrative of algorithms in linear regression, where plain gradient descent algorithms choose minimum $\ell_2$-norm solutions, and $ii$ there is recent work that purposely regularizes training algorithms towards minimum norm solutions [Bansal et al. (2018)].

Our findings support our claims: in particular, for the case of MobileNet and MobileNetV2, Adam, an adaptive method, converges to a solution that has at least as good generalization as plain gradient methods, while having $2\times$ larger $\ell_2$-norm weights. However, this may not always be the trend: in Figure 2 left panel, the plain gradient descent models for the PreActResNet18 architecture [He et al. (2016b)] show slightly better performance, while preserving low weight norm. The same holds also for the case of GoogleNet; see Figure 2 right panel.

6 Conclusions and Future Work

In this work, we re-visited the question of how adaptive and non-adaptive training algorithms compare: $i$ focusing on the linear regression setting as in [Wilson et al. (2017)], we show that there are similarities and differences between the behavior of adaptive and non-adaptive methods, depending on whether we have under- or over-parameterization; even when similarities may occur, there are differences on how the hyper-parameters are set between the two algorithmic classes, in order to obtain similar behavior. $ii$ In the over-parameterized linear regression case, we provide a small toy example showing that adaptive methods, such as AdaGrad, tend to generalize better than plain gradient descent, under assumptions; however, this is not a rule that applies universally. $iii$ Our findings on training DNNs show that there is no clear and provable superiority of plain or adaptive gradient methods. What was clear though from our experiments is that adaptive methods may converge to a model that has better generalization properties, while the $\ell_2$-norm of the weights is larger, but often require no less fine tuning than the plain gradient methods.

We note the small superiority of non-adaptive methods on some DNN simulations is not fully understood, and needs further investigation, beyond the simple linear regression model. A preliminary analysis of regularization for over-parameterized linear regression reveals that it can act as an equalizer over the set of adaptive and non-adaptive optimization methods, i.e. force all optimizers to converge to the same solution.
However, more work is needed to analyze its effect on the overall generalization guarantees both theoretically and experimentally as compared to the non-regularized versions of these algorithms.

References

Bansal, Y., Advani, M., Cox, D., and Saxe, A. Minnorm training: an algorithm for training over-parameterized deep neural networks. 2018.

Belkin, M., Ma, S., and Mandal, S. To understand deep learning we need to understand kernel learning. arXiv preprint arXiv:1802.01396, 2018.

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In Neural networks: Tricks of the trade, pp. 437–478. Springer, 2012.

Blum, A. and Rivest, R. Training a 3-node neural network is NP-complete. In Advances in neural information processing systems, pp. 494–501, 1989.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT’2010, pp. 177–186. Springer, 2010.

Bottou, L. Stochastic gradient descent tricks. In Neural networks: Tricks of the trade, pp. 421–436. Springer, 2012.

Dowler, D. Bounding the norm of matrix powers. 2013.

Dozat, T. Incorporating Nesterov momentum into Adam. 2016.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159, 2011.

Gunasekar, S., Lee, J., Soudry, D., and Srebro, N. Characterizing implicit bias in terms of optimization geometry. arXiv preprint arXiv:1802.08246, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In European Conference on Computer Vision, pp. 630–645. Springer, 2016b.

Horn, R. and Johnson, C. Matrix analysis. 1985.

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101, 2017.

McDonnell, M. D. and Vladusich, T. Enhanced image classification with a fast-learning shallow convolutional neural network. In Neural Networks (IJCNN), 2015 International Joint Conference on, pp. 1–7. IEEE, 2015.

Mukkamala, M. C. and Hein, M. Variants of RMSProp and AdaGrad with logarithmic regret bounds. arXiv preprint arXiv:1706.05507, 2017.

Nacson, M. S., Lee, J., Gunasekar, S., Srebro, N., and Soudry, D. Convergence of gradient descent on separable data. arXiv preprint arXiv:1803.01905, 2018.
Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Orr, G. and Müller, K.-R. *Neural networks: tricks of the trade*. Springer, 2003.

Poggio, T., Kawaguchi, K., Liao, Q., Miranda, B., Rosasco, L., Boix, X., Hidary, J., and Mhaskar, H. Theory of deep learning III: explaining the non-overfitting puzzle. *arXiv preprint arXiv:1801.00173*, 2017.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

Schaul, T., Zhang, S., and LeCun, Y. No more pesky learning rates. In *International Conference on Machine Learning*, pp. 343–351, 2013.

Senior, A., Heigold, G., and Yang, K. An empirical study of learning rates in deep neural networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6724–6728. IEEE, 2013.

Smith, S. L., Kindermans, P.-J., and Le, Q. V. Don’t decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *Cvpr*, 2015.

Telgarsky, M. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Tieleman, T. and Hinton, G. Lecture 6.5-RMSPro: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Ward, R., Wu, X., and Bottou, L. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

Wilson, A., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4151–4161, 2017.

Xu, W. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011.

Zeiler, M. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhang, J. and Mitliagkas, I. YELLOWFIN and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.
A Supplementary material

A.1 Unfolding gradient descent in the under-parameterized setting

Let us unfold this recursion, assuming that $w_0 = 0$:

\[
\begin{align*}
    w_1 &= w_0 - \eta \nabla f(w_0) \\
    &= \eta X^\top y \\
    w_2 &= w_1 - \eta \nabla f(w_1) \\
    &= 2\eta X^\top y - \eta^2 (X^\top X)X^\top y \\
    &\vdots \\
    w_5 &= w_4 - \eta \nabla f(w_4) \\
    &= 5\eta X^\top y - 10\eta^2 (X^\top X)X^\top y + 10\eta^3 (X^\top X)^2X^\top y \\
    &\quad - 5\eta^4 (X^\top X)^3X^\top y + \eta^5 (X^\top X)^4X^\top y \\
    &\vdots 
\end{align*}
\]

What we observe is that:

- The coefficients follow the Pascal triangle principle and can be easily expressed through binomial coefficients.
- The step size $\eta$ appears with increasing power coefficient, as well as the term $(X^\top X)$.
- There are some constant terms, $X^\top$ and $y$.

The above lead to the following generic characterization of the gradient descent recursion:

\[
w_K = \left( \sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (X^\top X)^{i-1} \right) X^\top y
\]

The expression in the parentheses satisfies:

\[
\begin{align*}
    \sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (X^\top X)^{i-1} \\
    &= \sum_{i=1}^{K} (-1)^{i} \binom{K}{i} \cdot \eta^i \cdot (X^\top X)^{i-1} (X^\top X)^i \\
    &= (-X^\top X)^{-1} \cdot \sum_{i=1}^{K} (-1)^{i} \binom{K}{i} \cdot \eta^i \cdot (X^\top X)^i \\
    &= (-X^\top X)^{-1} \cdot \sum_{i=1}^{K} \binom{K}{i} \cdot (-\eta X^\top X)^i \\
    &= (-X^\top X)^{-1} \cdot \left( \sum_{i=0}^{K} \binom{K}{i} \cdot (-\eta X^\top X)^i - \binom{K}{0} \cdot (-\eta X^\top X)^0 \right) \\
    &= (-X^\top X)^{-1} \cdot \left( \sum_{i=0}^{K} \binom{K}{i} \cdot (-\eta X^\top X)^i - I \right) \\
    &= (-X^\top X)^{-1} \cdot \left( \sum_{i=0}^{K} \binom{K}{i} \cdot I^{K-i} \cdot (-\eta X^\top X)^i - I \right)
\end{align*}
\]

Since $I$ and $-\eta X^\top X$ commute, we can use the binomial theorem:

\[
\sum_{i=0}^{K} \binom{K}{i} \cdot I^{K-i} \cdot (-\eta X^\top X)^i = (I - \eta X^\top X)^K.
\]
Thus, we finally get:

$$\sum_{i=1}^{K} (-1)^{i-1} \cdot \left(\frac{K}{i}\right) \cdot \eta^i \cdot (X^\top X)^{i-1} = (-X^\top X)^{-1} \cdot ((I - \eta X^\top X)^K - I)$$

### A.2 $D_k = D$ is a diagonal matrix with $D \succ 0$.

Here, we simplify the selection of preconditioner $D_k$ in adaptive gradient methods. Our purpose is to characterize their performance, and check how an adaptive (=preconditioned) algorithm performs in both under- and over-parameterized settings.

#### A.2.1 Under-parameterized linear regression.

Unfolding the “adaptive” gradient descent recursion for $w_0$, we get:

$$w_1 = w_0 - \eta D \nabla f(w_0) = \eta DX^\top y$$
$$w_2 = w_1 - \eta D \nabla f(w_1)$$
$$= 2\eta DX^\top y - \eta^2 D(X^\top XD)X^\top y$$
$$\vdots$$
$$w_5 = w_4 - \eta D \nabla f(w_4)$$
$$= 5\eta DX^\top y - 10\eta^2 D(X^\top XD)X^\top y$$
$$+ 10\eta^3 D(X^\top XD)^2X^\top y$$
$$- 5\eta^4 D(X^\top XD)^3X^\top y + \eta^5 D(X^\top XD)^4X^\top y,$$
$$\vdots$$

leading to the following closed form solution:

$$w_K = D \left( \sum_{i=1}^{K} (-1)^{i-1} \cdot \left(\frac{K}{i}\right) \cdot \eta^i \cdot (X^\top XD)^{i-1} \right) X^\top y.$$
Using Theorem 1, we can again prove that, for sufficiently large $K$ and for sufficiently small step size $\eta < \frac{1}{\lambda_1(X^\top XD)}$, we can prove that $\|I - \eta X^\top XD\| < 1$, and thus, $I - \eta X^\top XD \to 0$. Thus, for sufficiently large $K$:

$$w_K = (-X^\top X)^{-1} \cdot (-I) X^\top y = (X^\top X)^{-1} \cdot X^\top y,$$

which is the left inverse solution, as in gradient descent.

### A.2.2 Over-parameterized linear regression.

For the over-parameterized linear regression, we obtain a different expression by using a different kind of variable grouping in the unfolding procedure. In particular, we need to take in consideration that now $XX^\top$ is full rank, and thus the matrix $XDX^\top$ is also full rank, and thus invertible. Going back to the main preconditioned gradient descent recursion:

$$w_1 = w_0 - \eta D \nabla f(w_0) = \eta DX^\top y$$
$$w_2 = w_1 - \eta D \nabla f(w_1) = 2\eta DX^\top y - \eta^2 DX^\top (XDX^\top)y$$
$$\vdots$$
$$w_5 = w_4 - \eta D \nabla f(w_4) = 5\eta DX^\top y - 10\eta^2 DX^\top (XDX^\top)y + 10\eta^3 DX^\top (XDX^\top)^2y - 5\eta^4 DX^\top (XDX^\top)^3y + \eta^5 DX^\top (XDX^\top)^4y,$$

leading to the following closed form solution:

$$w_K = DX^\top \left( \sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (XDX^\top)^{i-1} \right) y.$$

The sum can be similarly simplified as:

$$\sum_{i=1}^{K} (-1)^{i-1} \cdot \binom{K}{i} \cdot \eta^i \cdot (XDX^\top)^{i-1} = (-XDX^\top)^{-1} \cdot \left( (I - \eta XDX^\top)^K - I \right)$$

This further transforms our recursion into:

$$w_K = DX^\top (-XDX^\top)^{-1} \cdot \left( (I - \eta XDX^\top)^K - I \right) y$$

Using Theorem 1, we can again prove that, for sufficiently large $K$ and for sufficiently small step size $\eta < \frac{1}{\lambda_1(XDX^\top)}$, we can prove that $\|I - \eta XDX^\top\| < 1$, and thus, $I - \eta XDX^\top \to 0$. Thus, for sufficiently large $K$:

$$w_K = DX^\top (-XDX^\top)^{-1} \cdot (-I) y = DX^\top (XDX^\top)^{-1} y \neq w_{mn}$$

which is not the same as the minimum norm solution, except when $D = \alpha I$ for some constant $\alpha > 0$. This proves that preconditioned algorithms might lead to different solutions, depending on the selection of the preconditioning matrix/matrices.
A.3 Unfolding adaptive gradient descent with varying $D_k$ in the under-parameterized setting

Unfolding the recursion, when $D_k$ is varying, we get:

\[ w_1 = w_0 - \eta D_0 \nabla f(w_0) = \eta D_0 X^\top y \]
\[ w_2 = w_1 - \eta D_1 \nabla f(w_1) = \eta (D_0 + D_1) X^\top y - \eta^2 (D_1 X^\top X D_0) X^\top y \]
\[ w_3 = w_2 - \eta D_2 \nabla f(w_2) = \eta (D_0 + D_1 + D_2) X^\top y - \eta^2 \left( D_1 X^\top X D_0 + D_2 X^\top X (D_0 + D_1) \right) X^\top y + \eta^3 D_2 X^\top X D_1 X^\top X D_0 X^\top y \]
\[ w_4 = w_3 - \eta D_3 \nabla f(w_3) \]
\[ = \eta \left( \sum_{i=0}^{3} D_i \right) X^\top y - \eta^2 \left( D_3 X^\top X \left( \sum_{i=0}^{2} D_i \right) + D_2 \left( X^\top X \sum_{i=0}^{1} D_i \right) + D_1 \left( X^\top X D_0 \right) \right) X^\top y + \eta^3 \left( D_3 (X^\top X) D_1 (X^\top X) D_0 + D_3 (X^\top X) D_2 (X^\top X) (D_1 + D_0) + D_2 (X^\top X) D_1 (X^\top X) D_0 \right) X^\top y \]
\[ w_5 = w_4 - \eta D_4 \nabla f(w_4) \]
\[ = \eta \left( \sum_{i=0}^{4} D_i \right) X^\top y - \eta^2 \left( D_4 H \left( \sum_{i=0}^{3} D_i \right) + D_3 H \left( \sum_{i=0}^{2} D_i \right) + D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_1 H D_0 \right) X^\top y + \eta^3 \left( D_4 H D_3 H \left( \sum_{i=0}^{2} D_i \right) + D_4 H D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_4 H D_1 H D_0 \right) X^\top y + \eta^4 D_3 H D_2 H D_1 H D_0 X^\top y \]
\[ w_6 = \ldots \]

where $H := X^\top X$.

A.4 Proof of Proposition 1

We will prove this proposition by induction.

**Base case**: Here, we compute the first iteration, $K = 1$:

\[ w_1 = (-X^\top X)^{-1} \cdot \left( \prod_{i=0}^{0} \left( I - \eta X^\top X D_i \right) - I \right) X^\top y \]
\[ = (-X^\top X)^{-1} \cdot \left( I - \eta X^\top X D_0 \right) X^\top y = \eta D_0 X^\top y \]

where we abuse the notation $\prod_{i=0}^{0} A_i = A_0$. This is the same result as in unfolding the recursion for $k = 0$, and assuming $w_0 = 0$. 

16
We will prove this proposition using induction.

\[ w_{K-1} = \left(-X^\top X\right)^{-1} \left( \prod_{i=K+2}^0 \left( I - \eta X^\top XD_i \right) - I \right) X^\top y. \]

Here, we use the convention \( \prod_{i=\alpha}^\beta A_i = A_\alpha \cdot A_{\alpha-1} \cdots A_{\beta+1} \cdot A_\beta \), for integers \( \alpha > \beta \). Then, the expression at the \( K \)-iteration satisfies:

\[
w_K = w_{K-1} - \eta D_{K-1} \nabla f(w_{K-1}) = w_{K-1} - \eta D_{K-1} X^\top (X w_{K-1} - y)
= \left( I - \eta D_{K-1} X^\top X \right) w_{K-1} + \eta D_{K-1} X^\top y
\quad \overset{(i)}{=} \left( I - \eta D_{K-1} X^\top X \right) \left(-X^\top X\right)^{-1} \left( \prod_{i=K+2}^0 \left( I - \eta X^\top XD_i \right) - I \right) X^\top y + \eta D_{K-1} X^\top y
= \left( \left(-X^\top X\right)^{-1} + \eta D_{K-1} \right) \left( \prod_{i=K+2}^0 \left( I - \eta X^\top XD_i \right) \right) X^\top y
\quad \overset{(i)}{=} \left( \left(-X^\top X\right)^{-1} + \eta D_{K-1} \right) X^\top y + \eta D_{K-1} X^\top y
= \left( \left(-X^\top X\right)^{-1} + \eta D_{K-1} \right) \left( \prod_{i=K+2}^0 \left( I - \eta X^\top XD_i \right) \right) X^\top y - \left(-X^\top X\right)^{-1} X^\top y
= \left(-X^\top X\right)^{-1} \left( I - \eta X^\top XD_{K-1} \right) \left( \prod_{i=K+2}^0 \left( I - \eta X^\top XD_i \right) \right) X^\top y - \left(-X^\top X\right)^{-1} X^\top y
= \left(-X^\top X\right)^{-1} \left( \prod_{i=K+1}^0 \left( I - \eta X^\top XD_i \right) \right) X^\top y - \left(-X^\top X\right)^{-1} X^\top y
= \left(-X^\top X\right)^{-1} \left( \prod_{i=K+1}^0 \left( I - \eta X^\top XD_i \right) - I \right) X^\top y
\]

where (i) is due to the inductive assumption. This completes the proof.

### A.5 Proof of Proposition 2

We will prove this proposition using induction.

**Base case:** Here, we compute the first iteration, \( K = 1 \):

\[
\hat{y}_1 = - \left( \prod_{i=0}^0 \left( I - \eta XD_i X^\top \right) - I \right) y = - \left( I - \eta XD_0 X^\top - I \right) y = \eta XD_0 X^\top y
\]

where, once again, we abuse the notation \( \prod_{i=0}^0 A_i = A_0 \). This is the same result as in unfolding the recursion for \( k = 0 \) above, and assuming \( w_0 = 0 \).

**Inductive case:** Assume the following is true

\[
\hat{y}_{K-1} = - \left( \prod_{i=K-2}^0 \left( I - \eta XD_i X^\top \right) - I \right) y = Xw_{K-1}
\]
Then,
\[ \hat{y}_K = X w_K = X w_{K-1} - \eta X D_{K-1} X^T (X w_{K-1} - y) \]
\[ = (I - \eta X D_{K-1} X^T) X w_{K-1} + \eta X D_{K-1} X^T y \]
\[ = -(I - \eta X D_{K-1} X^T) \left( \prod_{i=K-2}^{0} (I - \eta X D_i X^T) - I \right) y + \eta X D_{K-1} X^T y \]
\[ = - \left( \prod_{i=K-1}^{0} (I - \eta X D_i X^T) - I \right) y \]

### A.6 Proof of Proposition 3

We will prove this using induction. Let \( A = \text{diag}(|X^T y|) \), \( Q = \text{diag}(|X^T y|^3) = A^3 \). The inversion is possible since we assume that \( X^T y \) have all components that are not zero. We will show that

\[ w_k = \lambda_k Q^{-1} \text{sign}(X^T y) = A^{-3} \text{sign}(X^T y), \]

for some \( \lambda_k \). \( w_0 = 0 \) is satisfied for \( \lambda_0 = 0 \), and thus the base case is trivially true. Now, we will assume the assertion holds for all iterations up to \( k \). Observe that the gradient at the \( k \)-th iteration satisfies:

\[ \nabla f(w_k) = X^T (X w_k - y) = \lambda_k X^T X Q^{-1} \text{sign}(X^T y) - X^T y = (\lambda_k c - 1) X^T y \]

where the last inequality follows from the assumption that \( XQ^{-1}\text{sign}(X^T y) = cy \). Let us define \( H_k \) the denominator of in the preconditioner \( D_k \); for simplicity, we assume \( \varepsilon \) a very small, negligible quantity. Then, we have:

\[ H_k = \text{diag} \left( \left( \sum_{s=1}^{k} \nabla f(w_s) \odot \nabla f(w_s) \right)^2 \right) \propto \text{diag} \left( \left( \sum_{s=1}^{k} (\lambda_s c - 1) \right)^2 |X^T y|^4 \right) = \nu_k \cdot \text{diag}(|X^T y|^4) = \nu_k A^4, \]

for some constant \( \nu_k \). Then, focusing on the \( k+1 \)-th iteration, we get:

\[ w_{k+1} = w_k - \eta_k H_k^{-1} X^T (X w_k - y) \]
\[ = w_k - \eta_k H_k^{-1} X^T X w_k + \eta_k H_k^{-1} X^T y \]
\[ \overset{(i)}{=} \lambda_k Q^{-1} \text{sign}(X^T y) - \lambda_k \eta_k H_k^{-1} X^T X Q^{-1} \text{sign}(X^T y) + \eta_k H_k^{-1} X^T y \]
\[ = \lambda_k Q^{-1} \text{sign}(X^T y) - \lambda_k \alpha_k c H_k^{-1} X^T y + \alpha_4 H_k^{-1} X^T y \]
\[ \overset{(ii)}{=} \lambda_k A^{-3} \text{sign}(X^T y) - \frac{\lambda_k \alpha_k c}{\nu_k} A^{-4} \text{diag}(|X^T y|) \text{sign}(X^T y) + \frac{\alpha_k}{\nu_k} A^{-4} \text{diag}(|X^T y|) \text{sign}(X^T y) \]
\[ \overset{(iii)}{=} \lambda_k A^{-3} \text{sign}(X^T y) - \frac{\lambda_k \alpha_k c}{\nu_k} A^{-3} \text{sign}(X^T y) + \frac{\alpha_k}{\nu_k} A^{-3} \text{sign}(X^T y) \]
\[ \overset{(iv)}{=} \left( \lambda_k - \frac{\lambda_k \alpha_k c}{\nu_k} + \frac{\alpha_k}{\nu_k} \right) Q^{-1} \text{sign}(X^T y) \]
\[ = \lambda_{k+1} Q^{-1} \text{sign}(X^T y) \]

where (i) is due to the assumption \( w_k = \lambda_k Q^{-1} \text{sign}(X^T y) \) and equality (ii) follows from \( v = \text{diag}(|v|) \text{sign}(v) \) and equations (iii) and (iv). (iii) results from the definition of \( A \) and (iv) holds by the definition of \( Q^{-1} = A^{-3} \).

### A.7 More results on the counterexample

We first provide the same table in Table 1 but with unnormalized values for distances with respect to Adagrad variant.

Here, we provide further results on the counterexample in Subsubsection 4.3. Tables 4 and 5 contains results for \( J = 10 \); the purpose of these tables is to show that even if we change the memory use footprint of
Table 3: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and adaptive gradient descent methods. We set \( p = 7/8 \) and \( J = 10 \), as in the main text. The adaptive method uses \( D_k \) according to (2). The distances shown are median values out of 100 different realizations for each setting; the accuracies are obtained by testing \( 10^4 \) predictions on unseen data.

| \( n \) | \( \ell \) | \( D \) | \( w_0 - w_{mn} \) | \( \| w_0 - w_{mn} \|_2 \) | Gradient Descent | AdaGrad variant | Adam |
|---|---|---|---|---|---|---|---|
| 10 | 1/32 | Acc. (%) | 63 | 100 | 91 | 1.015 \( \cdot 10^{-16} \) | 0.9911 | 0.1007 |
| 1/16 | Acc. (%) | 53 | 100 | 87 | 1.7401 \( \cdot 10^{-16} \) | 0.9263 | 0.0864 |
| 1/8 | Acc. (%) | 58 | 99 | 84 | 4.08 \( \cdot 10^{-16} \) | 0.8179 | 0.0764 |
| 50 | 1/32 | Acc. (%) | 77 | 100 | 88 | 4.729 \( \cdot 10^{-15} \) | 0.8893 | 0.0271 |
| 1/16 | Acc. (%) | 80 | 100 | 89 | 6.9197 \( \cdot 10^{-15} \) | 0.7929 | 0.0628 |
| 1/8 | Acc. (%) | 91 | 100 | 89 | 9.7170 \( \cdot 10^{-15} \) | 0.6639 | 0.1767 |
| 100 | 1/32 | Acc. (%) | 85 | 100 | 95 | 4.975 \( \cdot 10^{-9} \) | 0.8463 | 0.0344 |
| 1/16 | Acc. (%) | 83 | 100 | 76 | 2.5420 \( \cdot 10^{-9} \) | 0.7217 | 0.1020 |
| 1/8 | Acc. (%) | 100 | 100 | 90 | 1.5572 \( \cdot 10^{-11} \) | 0.6289 | 0.3306 |

Table 4: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and adaptive gradient descent methods. We set \( p = 3/8 \) and \( J = 10 \), as in the main text. The adaptive method uses \( D_k \) according to (2). The distances shown are median values out of 100 different realizations for each setting; the accuracies are obtained by testing \( 10^4 \) predictions on unseen data.

| \( n \) | \( \ell \) | \( D \) | \( w_0 - w_{mn} \) | \( \| w_0 - w_{mn} \|_2 \) | Gradient Descent | AdaGrad variant | RMSProp | Adam |
|---|---|---|---|---|---|---|---|---|
| 10 | 1/32 | Acc. (%) | 67 | 100 | 60 | 60 | 55 | 1.015 \( \cdot 10^{-16} \) | 0.9911 | 0.0965 | 0.1007 |
| 1/16 | Acc. (%) | 67 | 100 | 61 | 1.7401 \( \cdot 10^{-16} \) | 0.9263 | 0.7221 | 0.0864 |
| 1/8 | Acc. (%) | 60 | 86 | 58 | 4.08 \( \cdot 10^{-16} \) | 0.8179 | 1.1454 | 0.0764 |
| 50 | 1/32 | Acc. (%) | 56 | 93 | 56 | 4.729 \( \cdot 10^{-15} \) | 0.8893 | 0.2707 | 0.0271 |
| 1/16 | Acc. (%) | 63 | 92 | 63 | 6.9197 \( \cdot 10^{-15} \) | 0.7929 | 1.6255 | 0.0628 |
| 1/8 | Acc. (%) | 53 | 70 | 53 | 9.7170 \( \cdot 10^{-15} \) | 0.6639 | 12.6065 | 0.1767 |
| 100 | 1/32 | Acc. (%) | 66 | 97 | 66 | 4.975 \( \cdot 10^{-9} \) | 0.8463 | 0.3827 | 0.0344 |
| 1/16 | Acc. (%) | 61 | 87 | 61 | 2.5420 \( \cdot 10^{-9} \) | 0.7217 | 2.3068 | 0.1020 |
| 1/8 | Acc. (%) | 65 | 74 | 65 | 1.5572 \( \cdot 10^{-11} \) | 0.6289 | 21.7775 | 0.3306 |

The AdaGrad variant—by storing fewer or more gradients to compute \( D_k \) in (2)—the results are the same: the AdaGrad variant consistently converges to a solution different than the minimum norm solution, while being more accurate than the latter for small values of \( \ell \) (i.e., smaller margin between the two classes).
when the levels are around 1, but the superiority (or the lack of inferiority) of adaptive methods is there to

table 5: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and

variant, Adam and RMSProp. Using these plots, our aim is to gain insight on how the generalization

and test datasets have no overlap. We include this in comparison for completeness.

features across classes:

Ignoring the features that are zero everywhere, the counter-example described in (1) has three main types of

features across classes:

common features (c): The features which are identical irrespective of the class they belong to.

distinguishing feature (d): The features which are different for each class, and, in some sense, are the

defining characteristic of the class.

We note that using only the three elements in adaptive methods is not backed up by theory since it assumes that the training

and test datasets have no overlap. We include this in comparison for completeness.

Table 5: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and

adaptive gradient descent methods. We set \( p = 5/8 \) and \( J = 10 \), as in the main text. The adaptive method

uses \( D_k \) according to (2). The distances shown are median values out of 100 different realizations for each

setting; the accuracies are from \( 10^4 \) predictions on unseen data.

In Tables 4-6, the distances reported for adagrad variant are calculated after normalizing the converged \( \hat{w} \). Normalization of outputs does not affect the final prediction. Gradient descent, RMSprop and Adam are the usual distances from the converged value.

Plain gradient descent methods provably need to rely on the first elements to decide; using the same

rule for adaptive methods. The remaining subsection considers the case where we decide based on the

\( y = \text{sign}(x^\top w) \) rule, where \( w \) is the complete learned model. As we show empirically, more often than not

adaptive methods outperform plain gradient methods.

Observing the performance of various optimization techniques for different values of \( n, p \) and \( \ell \), we

observed that the best performances are obtained when the dataset is highly imbalanced irrespective of the

optimization algorithm chosen. When the data is (almost) balanced, it is difficult to comment on how the

performance of these algorithms is affected by variations in the levels \( \ell \) and probability \( p \).

Let us first describe the experimental setup. We consider the same counter-example in (1). We fix \( p \) and

compare the accuracy for different algorithms by varying the levels for each of gradient descent, AdaGrad

variant, Adam and RMSProp. Using these plots, our aim is to gain insight on how the generalization performance fluctuates. In Figure ?? \((n = 100)\), we observe that gradient descent performs relatively better when the levels are around 1, but the superiority (or the lack of inferiority) of adaptive methods is there to see for the remaining range of level values.

A.7.1 Interpreting the class of counterexamples

Ignoring the features that are zero everywhere, the counter-example described in (1) has three main types of

features across classes:

- Distinguishing feature (d): The features which are different for each class, and, in some sense, are the

  defining characteristic of the class.

- Common features (c): The features which are identical irrespective of the class they belong to.
Unique features (u): Features which are non-zero at particular coordinates for only that counter-example, and zero for others.

The first coordinate belongs to class ‘d’, the second and third coordinates belong to class ‘c’ and the remaining coordinates either belong to class ‘u’ or are zero everywhere in the counter-example given by (1). Changing the number of one or more of these features has little to no effect on the claims we make in this paper.

To further illustrate that the results obtained on the counterexample [1] are not one-off, and the inconclusive behavior is seen across a wide variety of examples, we create a similar counter-example which has one feature of type ‘d’, four features of type ‘c’ and one feature of type ‘u’:

\[
(X_i)_j = \begin{cases} 
    y_i, & j = 1, \\
    1, & j = 2, 3, 4, 6 \\
    1, & j = 5 + 5(i - 1), \\
    0, & \text{otherwise.}
\end{cases}
\] if \( y_i = 1 \),

\[
(X_i)_j = \begin{cases} 
    y_i, & j = 1, \\
    1, & j = 2, 3, 4, 6 \\
    1, & j = 7 + 5(i - 1), \\
    0, & \text{otherwise.}
\end{cases}
\] if \( y_i = -1 \). (5)

We will once again evaluate the performance of all four optimization algorithms for \( n = 50 \), over different values of \( \ell \) and \( p \). In Table 6, it is once again impossible to find a clearly dominant optimization approach, assuming we use the \( y = \text{sign}(x^\top w) \) rule.

| \( \ell \) | \( p \) | Acc. (%) | Gradient Descent | AdaGrad variant | RMSProp | Adam |
|----------|-------|---------|-----------------|----------------|---------|------|
| 0.1      | 0.3/8 | 70      | 53              | 70             | 70      | 70   |
|          | 0.5/8 | 41      | 36              | 30             | 26      |      |
| 0.2      | 0.3/8 | 56      | 45              | 39             | 36      |      |
|          | 0.5/8 | 49      | 69              | 50             | 58      |      |
| 0.5      | 0.3/8 | 57      | 52              | 50             | 60      | 53   |
|          | 0.5/8 | 53      | 50              | 60             | 53      |      |
| 1        | 0.3/8 | 54      | 70              | 60             | 54      |      |
|          | 0.5/8 | 52      | 50              | 48             | 50      | 55   |
| 2        | 0.3/8 | 59      | 70              | 40             | 59      |      |
|          | 0.5/8 | 53      | 60              | 30             | 53      |      |
| 5        | 0.3/8 | 57      | 70              | 70             | 57      |      |
|          | 0.5/8 | 56      | 62              | 62             | 56      |      |
| 10       | 0.3/8 | 67      | 60              | 60             | 67      |      |
|          | 0.5/8 | 52      | 46              | 46             | 52      |      |
| 20       | 0.3/8 | 51      | 40              | 40             | 51      |      |
|          | 0.5/8 | 54      | 62              | 62             | 54      |      |

Table 6: Prediction accuracy and distances from the minimum norm solution for plain gradient descent and adaptive gradient descent methods for the new counter-example. We set \( n = 50 \) and \( J = 10 \), as in the main text. The adaptive method uses \( D_k \) according to (2). The distances shown are median values out of 10 different realizations for each setting; the accuracies are from \( 10^2 \) predictions on unseen data.
A.8 Hyperparameter tuning

Tuning the hyperparameters is a crucial step in extracting the best performance out of deep neural networks. In this paper, we adopt a grid based approach for tuning hyperparameters.

| Hyperparameters | Tuning Set |
|-----------------|------------|
| Stepsizes       | {0.0001, 0.001, 0.01, 0.1, 1} |
| Momentum        | {0.0, 0.25, 0.5, 0.75, 0.9} |

Table 7: This table lists the set of hyperparameters used for tuning various algorithms.

Decay-based hyperparameter tuning achieved significant gains in non-adaptive optimizers thus justifying their utility. The performance of adaptive optimizers was largely left unchanged by decay based methods due to the inherent decay in these optimization algorithms.

| Decay Type | Parameters | Tuning Set |
|------------|------------|------------|
| Step       | Decay Rate | {0.1, 0.5, 0.8, 0.9} |
|            | Decay Step | {0, 10, 20, 40, 80, 160} |

Table 8: This table lists the decay approaches used for tuning various algorithms.

Developmental decay was not used for MNIST, CIFAR-10 datasets since they do not possess any validation set. Testing on artificially generated developmental (validation) sets will hinder fair comparison and understanding of the optimization algorithms.

A.9 $\ell_2$-norm regularization: The great equalizer for adaptive and non-adaptive methods

Wilson et al. (2017) shows that for the over-parameterized setting non-adaptive methods perform as well as or better than their adaptive counterparts. In this paper, we try to show that this is not necessarily the case and it is indeed possible to obtain significant gains using adaptive methods depending on the problem at hand. In this subsection, we show that the adaptive and non-adaptive methods converge to the same unique solution in Ridge regularization ($\ell_2$-norm regularization).

Let us consider the regularized version of the linear regression problem:

$$\min_w f(w; \lambda) = \min_w \|Xw - y\|^2_2 + \lambda\|w\|^2_2$$

We know that non-adaptive gradient based methods converge to the following solution: $w = (X^TX + \lambda I)^{-1}X^Ty$.

What about non-adaptive methods? In this subsection, we take a look at all the adaptive methods whose updates satisfy $w_{k+1} = w_k - \eta D_k \nabla f(w_k; \lambda)$. Adagrad, Adadelta, RMSProp all fall under this framework.

**Proposition 4** Consider the over-parameterized case in ridge regression. Assume the recursion $w_{k+1} = w_k - \eta D_k \nabla f(w_k; \lambda)$ Then, after $K$ iterations, $w_K$ satisfies:

$$w_K = \left( -X^TX + \lambda I \right)^{-1} \cdot \left( \prod_{i=K-1}^0 \left( I - \eta \left( X^TX + \lambda I \right) D_i \right) - I \right) X^Ty.$$

**Proof:**
Let $H = X^\top X + \lambda I$

$$w_1 = w_0 - \eta D_0 \nabla f(w_0; \lambda) = w_0 - \eta D_0 (X^\top (Xw_0 - y) + \lambda w_0) = \eta D_0 X^\top y$$

$$w_2 = w_1 - \eta D_1 \nabla f(w_1; \lambda) = w_1 - \eta D_1 (X^\top (Xw_1 - y) + \lambda w_1) = \eta(D_0 + D_1) X^\top y - \eta^2 D_1 (X^\top X + \lambda I) D_0 X^\top y$$

$$w_3 = w_2 - \eta D_2 \nabla f(w_2; \lambda) = \eta(D_0 + D_1 + D_2) X^\top y - \eta^2 (D_1 (X^\top X + \lambda I) D_0 + D_2 (X^\top X + \lambda I) (D_0 + D_1)) X^\top y$$

$$+ \eta^3 D_2 (X^\top X + \lambda I) D_1 (X^\top X + \lambda I) D_0 X^\top y$$

$$w_4 = w_3 - \eta D_3 \nabla f(w_3; \lambda) = \eta \left( \sum_{i=0}^{3} D_i \right) X^\top y$$

$$- \eta^2 \left( D_3 (X^\top X + \lambda I) \cdot \left( \sum_{i=0}^{2} D_i \right) + D_2 (X^\top X + \lambda I) \left( \sum_{i=0}^{1} D_i \right) + D_1 (X^\top X + \lambda I) D_0 \right) X^\top y$$

$$+ \eta^3 \left( D_3 (X^\top X + \lambda I) D_1 (X^\top X + \lambda I) D_0 + D_2 (X^\top X + \lambda I) D_2 (X^\top X + \lambda I) (D_0 + D_1) \right) X^\top y$$

$$- \eta^4 D_3 (X^\top X + \lambda I) D_2 (X^\top X + \lambda I) D_1 (X^\top X + \lambda I) D_0 X^\top y$$

$$w_5 = w_4 - \eta D_4 \nabla f(w_4; \lambda)$$

$$= \eta \left( \sum_{i=0}^{4} D_i \right) X^\top y$$

$$- \eta^2 \left( D_4 H \left( \sum_{i=0}^{3} D_i \right) + D_3 H \left( \sum_{i=0}^{2} D_i \right) + D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_1 H D_0 \right)$$

$$+ \eta^3 \left( D_4 H D_3 H \left( \sum_{i=0}^{2} D_i \right) + D_4 H D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_4 H D_1 H D_0 \right)$$

$$+ \eta^4 \left( D_4 H D_3 H D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_4 H D_3 H D_1 H D_0 \right)$$

$$X^\top y$$

$$- \eta^4 \left( D_4 H D_3 H D_2 H \left( \sum_{i=0}^{1} D_i \right) + D_4 H D_3 H D_1 H D_0 \right)$$

$$+ \eta^5 D_4 H D_3 H D_2 H D_1 H D_0 X^\top y$$

**Base case:** Here, we compute the first iteration, $K = 1$:

$$w_1 = -(X^\top X + \lambda I)^{-1} \cdot \left( \prod_{i=0}^{a} \left( I - \eta (X^\top X + \lambda I) D_i \right) - I \right) X^\top y$$

$$= -(X^\top X + \lambda I)^{-1} \cdot \left( \left( I - \eta (X^\top X + \lambda I) D_0 \right) - I \right) X^\top y = \eta D_0 X^\top y$$

where we abuse the notation $\prod_{i=0}^{a} A_i = A_0$. This is the same result as in unfolding the recursion for $k = 0$, and assuming $w_0 = 0$.\

\[23\]
**Inductive case:** Now, assume that, the above statement holds for $K - 1$,

$$w_{K-1} = -\left(X^\top X + \lambda I\right)^{-1} \cdot \left(\prod_{i=K-2}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right) - I\right) X^\top y.$$  

Here, we use the convention $\prod_{i=\alpha}^{\beta} A_i = A_\alpha \cdot A_{\alpha+1} \cdots A_{\beta+1} \cdot A_\beta$, for integers $\alpha > \beta$. Then, the expression at the $K$-iteration satisfies:

$$w_K = w_{K-1} - \eta D_{K-1} \nabla f(w_{K-1}) = w_{K-1} - \eta D_{K-1} X^\top (X w_{K-1} - y)$$

$$= \left(I - \eta D_{K-1} (X^\top X + \lambda I)\right) w_{K-1} + \eta D_{K-1} X^\top y$$

$$w_K \overset{(i)}{=} \left(I - \eta D_{K-1} (X^\top X + \lambda I)\right) \left(-(X^\top X + \lambda I)\right)^{-1} \cdot \left(\prod_{i=K-2}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right) - I\right) X^\top y$$

$$+ \eta D_{K-1} X^\top y$$

$$= \left(-(X^\top X + \lambda I)\right)^{-1} + \eta D_{K-1} \cdot \left(\prod_{i=K-2}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right) - I\right) X^\top y + \eta D_{K-1} X^\top y$$

$$= \left(-(X^\top X + \lambda I)\right)^{-1} + \eta D_{K-1} \cdot \left(\prod_{i=K-2}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right)\right) X^\top y$$

$$- \left(-(X^\top X + \lambda I)\right)^{-1} X^\top y$$

$$= \left(-(X^\top X + \lambda I)\right)^{-1} \left(I - \eta(X^\top X + \lambda I)D_{K-1}\right) \cdot \left(\prod_{i=K-2}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right)\right) X^\top y$$

$$- \left(-(X^\top X + \lambda I)\right)^{-1} X^\top y$$

$$= \left(-(X^\top X + \lambda I)\right)^{-1} \cdot \left(\prod_{i=K-1}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right)\right) X^\top y - \left(-(X^\top X + \lambda I)\right)^{-1} X^\top y$$

$$= \left(-(X^\top X + \lambda I)\right)^{-1} \cdot \left(\prod_{i=K-1}^{0} \left(I - \eta(X^\top X + \lambda I)D_i\right) - I\right) X^\top y.$$

where (i) is due to the inductive assumption. This completes the proof.

Using Theorem [1], we can again infer that, for sufficiently large $K$ and for sufficiently small $\eta < \max_i \frac{1}{\lambda_i (X^\top X + \lambda I) D_i}$, such that $\|I - \eta(X^\top X + \lambda I)D_i\| < 1 \forall i$, we have: $\prod_{i=1}^{T-1} \left(I - \eta(X^\top X + \lambda I)D_i\right) \to 0$.

The aim here is not to comment on how the regularization affects the final solution but to show that with $\ell_2$-norm regularization, both adaptive and non-adaptive optimization algorithms converge to the same optimum. Loshchilov & Hutter [2017] demonstrates that $\ell_2$-norm regularization is not the same as weight decay for adaptive methods but it is equivalent to weight-decay for SGD.