On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization

Sanjeev Arora [1 2] Nadav Cohen [2] Elad Hazan [1 3]

Abstract

Conventional wisdom in deep learning states that increasing depth improves expressiveness but complicates optimization. This paper suggests that, sometimes, increasing depth can speed up optimization. The effect of depth on optimization is decoupled from expressiveness by focusing on settings where additional layers amount to overparameterization – linear neural networks, a well-studied model. Theoretical analysis, as well as experiments, show that here depth acts as a preconditioner which may accelerate convergence. Even on simple convex problems such as linear regression with ℓ_p loss, $p > 2$, gradient descent can benefit from transitioning to a non-convex overparameterized objective, more than it would from some common acceleration schemes. We also prove that it is mathematically impossible to obtain the acceleration effect of overparametrization via gradients of any regularizer.

1. Introduction

How does depth help? This central question of deep learning still eludes full theoretical understanding. The general consensus is that there is a trade-off: increasing depth improves expressiveness, but complicates optimization. Superior expressiveness of deeper networks, long suspected, is now confirmed by theory, albeit for fairly limited learning problems (Eldan & Shamir, 2015; Raghu et al., 2016; Lee et al., 2017; Cohen et al., 2017; Daniely, 2017; Arora et al., 2018). Difficulties in optimizing deeper networks have also been long clear – the signal held by a gradient gets buried as it propagates through many layers. This is known as the "vanishing/exploding gradient problem". Modern techniques such as batch normalization (Ioffe & Szegedy, 2015) and residual connections (He et al., 2015) have somewhat alleviated these difficulties in practice.

Given the longstanding consensus on expressiveness *vs.* optimization trade-offs, this paper conveys a rather counter-intuitive message: increasing depth can *accelerate* optimization. The effect is shown, via first-cut theoretical and empirical analyses, to resemble a combination of two well-known tools in the field of optimization: *momentum*, which led to provable acceleration bounds (Nesterov, 1983); and *adaptive regularization*, a more recent technique proven to accelerate by Duchi et al. (2011) in their proposal of the AdaGrad algorithm. Explicit mergers of both techniques are quite popular in deep learning (Kingma & Ba, 2014; Tieleman & Hinton, 2012). It is thus intriguing that merely introducing depth, with no other modification, can have a similar effect, but *implicitly*.

There is an obvious hurdle in isolating the effect of depth on optimization: if increasing depth leads to faster training on a given dataset, how can one tell whether the improvement arose from a true acceleration phenomenon, or simply due to better representational power (the shallower network was unable to attain the same training loss)? We respond to this hurdle by focusing on *linear neural networks* (*cf.* Saxe et al. (2013); Goodfellow et al. (2016); Hardt & Ma (2016); Kawaguchi (2016)). With these models, adding layers does not alter expressiveness; it manifests itself only in the replacement of a matrix parameter by a product of matrices – an *overparameterization*.

We provide a new analysis of linear neural network optimization via direct treatment of the differential equations associated with gradient descent when training arbitrarily deep networks on arbitrary loss functions. We find that the overparameterization introduced by depth leads gradient descent to operate as if it were training a shallow (single layer) network, while employing a particular preconditioning scheme. The preconditioning promotes movement along directions already taken by the optimization, and can be seen as an acceleration procedure that combines momentum with adaptive learning rates. Even on simple convex problems such as linear regression with ℓ_p loss, $p > 2$, overparameterization via depth can significantly speed up training. Surprisingly, in some of our experiments, not only did overparameterization outperform naïve gradient descent, but it was also faster than two well-known acceleration methods –

[1] Department of Computer Science, Princeton University, Princeton, NJ, USA [2] School of Mathematics, Institute for Advanced Study, Princeton, NJ, USA [3] Google Brain, USA. Correspondence to: Nadav Cohen <cohennadav@ias.edu>.

AdaGrad (Duchi et al., 2011) and AdaDelta (Zeiler, 2012). In addition to purely linear networks, we also demonstrate (empirically) the implicit acceleration of overparameterization on a non-linear model, by replacing hidden layers with depth-2 linear networks. The implicit acceleration of overparametrization is different from standard regularization – we prove its effect cannot be attained via gradients of *any* fixed regularizer.

Both our theoretical analysis and our empirical evaluation indicate that acceleration via overparameterization need not be computationally expensive. From an optimization perspective, overparameterizing using wide or narrow networks has the same effect – it is only the depth that matters.

The remainder of the paper is organized as follows. In Section 2 we review related work. Section 3 presents a warmup example of linear regression with ℓ_p loss, demonstrating the immense effect overparameterization can have on optimization, with as little as a single additional scalar. Our theoretical analysis begins in Section 4, with a setup of preliminary notation and terminology. Section 5 derives the preconditioning scheme implicitly induced by overparameterization, followed by Section 6 which shows that this form of preconditioning is not attainable via any regularizer. In Section 7 we qualitatively analyze a very simple learning problem, demonstrating how the preconditioning can speed up optimization. Our empirical evaluation is delivered in Section 8. Finally, Section 9 concludes.

2. Related Work

Theoretical study of optimization in deep learning is a highly active area of research. Works along this line typically analyze critical points (local minima, saddles) in the landscape of the training objective, either for linear networks (see for example Kawaguchi (2016); Hardt & Ma (2016) or Baldi & Hornik (1989) for a classic account), or for specific non-linear networks under different restrictive assumptions (*cf.* Choromanska et al. (2015); Haeffele & Vidal (2015); Soudry & Carmon (2016); Safran & Shamir (2017)). Other works characterize other aspects of objective landscapes, for example Safran & Shamir (2016) showed that under certain conditions a monotonically descending path from initialization to global optimum exists (in compliance with the empirical observations of Goodfellow et al. (2014)).

The dynamics of optimization was studied in Fukumizu (1998) and Saxe et al. (2013), for linear networks. Like ours, these works analyze gradient descent through its corresponding differential equations. Fukumizu (1998) focuses on linear regression with ℓ_2 loss, and does not consider the effect of varying depth – only a two (single hidden) layer network is analyzed. Saxe et al. (2013) also focuses on ℓ_2 regression, but considers any depth beyond two (inclusive), ultimately concluding that increasing depth can *slow down* optimization, albeit by a modest amount. In contrast to these two works, our analysis applies to a general loss function, and any depth including one. Intriguingly, we find that for ℓ_p regression, acceleration by depth is revealed only when $p > 2$. This explains why the conclusion reached in Saxe et al. (2013) differs from ours.

Turning to general optimization, accelerated gradient (momentum) methods were introduced in Nesterov (1983), and later studied in numerous works (see Wibisono et al. (2016) for a short review). Such methods effectively accumulate gradients throughout the entire optimization path, using the collected history to determine the step at a current point in time. Use of preconditioners to speed up optimization is also a well-known technique. Indeed, the classic Newton's method can be seen as preconditioning based on second derivatives. Adaptive preconditioning with only first-order (gradient) information was popularized by the BFGS algorithm and its variants (*cf.* Nocedal (1980)). Relevant theoretical guarantees, in the context of regret minimization, were given in Hazan et al. (2007); Duchi et al. (2011). In terms of combining momentum and adaptive preconditioning, Adam (Kingma & Ba, 2014) is a popular approach, particularly for optimization of deep networks.

Algorithms with certain theoretical guarantees for nonconvex optimization, and in particular for training deep neural networks, were recently suggested in various works, for example Ge et al. (2015); Agarwal et al. (2017); Carmon et al. (2016); Janzamin et al. (2015); Livni et al. (2014) and references therein. Since the focus of this paper lies on the analysis of algorithms already used by practitioners, such works lie outside our scope.

3. Warmup: ℓ_p Regression

We begin with a simple yet striking example of the effect being studied. For linear regression with ℓ_p loss, we will see how even the slightest overparameterization can have an immense effect on optimization. Specifically, we will see that simple gradient descent on an objective overparameterized by a single scalar, corresponds to a form of accelerated gradient descent on the original objective.

Consider the objective for a scalar linear regression problem with ℓ_p loss (p – even positive integer):

$$L(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim S}\left[\tfrac{1}{p}(\mathbf{x}^\top\mathbf{w} - y)^p\right]$$

$\mathbf{x} \in \mathbb{R}^d$ here are instances, $y \in \mathbb{R}$ are continuous labels, S is a finite collection of labeled instances (training set), and $\mathbf{w} \in \mathbb{R}^d$ is a learned parameter vector. Suppose now that we apply a simple overparameterization, replacing the parameter vector $\mathbf{w}$ by a vector $\mathbf{w}_1 \in \mathbb{R}^d$ times a scalar $w_2 \in \mathbb{R}$:

$$L(\mathbf{w}_1, w_2) = \mathbb{E}_{(\mathbf{x},y)\sim S}\left[\tfrac{1}{p}(\mathbf{x}^\top\mathbf{w}_1 w_2 - y)^p\right]$$

Obviously the overparameterization does not affect the expressiveness of the linear model. How does it affect optimization? What happens to gradient descent on this non-convex objective?

Observation 1. *Gradient descent over $L(\mathbf{w}_1, w_2)$, with fixed small learning rate and near-zero initialization, is equivalent to gradient descent over $L(\mathbf{w})$ with particular adaptive learning rate and momentum terms.*

To see this, consider the gradients of $L(\mathbf{w})$ and $L(\mathbf{w}_1, w_2)$:

$$\begin{aligned}\nabla_{\mathbf{w}} &:= \mathbb{E}_{(\mathbf{x},y)\sim S}\left[(\mathbf{x}^\top\mathbf{w} - y)^{p-1}\mathbf{x}\right]\\ \nabla_{\mathbf{w}_1} &:= \mathbb{E}_{(\mathbf{x},y)\sim S}\left[(\mathbf{x}^\top\mathbf{w}_1 w_2 - y)^{p-1}w_2\mathbf{x}\right]\\ \nabla_{w_2} &:= \mathbb{E}_{(\mathbf{x},y)\sim S}\left[(\mathbf{x}^\top\mathbf{w}_1 w_2 - y)^{p-1}\mathbf{w}_1^\top\mathbf{x}\right]\end{aligned}$$

Gradient descent over $L(\mathbf{w}_1, w_2)$ with learning rate $\eta > 0$:

$$\mathbf{w}_1^{(t+1)} \mapsfrom \mathbf{w}_1^{(t)} - \eta\nabla_{\mathbf{w}_1^{(t)}} \quad , \quad w_2^{(t+1)} \mapsfrom w_2^{(t)} - \eta\nabla_{w_2^{(t)}}$$

The dynamics of the underlying parameter $\mathbf{w} = \mathbf{w}_1 w_2$ are:

$$\begin{aligned}&\mathbf{w}^{(t+1)} = \mathbf{w}_1^{(t+1)} w_2^{(t+1)}\\ &\mapsfrom (\mathbf{w}_1^{(t)} - \eta\nabla_{\mathbf{w}_1^{(t)}})(w_2^{(t)} - \eta\nabla_{w_2^{(t)}})\\ &= \mathbf{w}_1^{(t)} w_2^{(t)} - \eta w_2^{(t)}\nabla_{\mathbf{w}_1^{(t)}} - \eta\nabla_{w_2^{(t)}}\mathbf{w}_1^{(t)} + \mathcal{O}(\eta^2)\\ &= \mathbf{w}^{(t)} - \eta(w_2^{(t)})^2\nabla_{\mathbf{w}^{(t)}} - \eta(w_2^{(t)})^{-1}\nabla_{w_2^{(t)}}\mathbf{w}^{(t)} + \mathcal{O}(\eta^2)\end{aligned}$$

η is assumed to be small, thus we neglect $\mathcal{O}(\eta^2)$. Denoting $\rho^{(t)} := \eta(w_2^{(t)})^2 \in \mathbb{R}$ and $\gamma^{(t)} := \eta(w_2^{(t)})^{-1}\nabla_{w_2^{(t)}} \in \mathbb{R}$, this gives:

$$\mathbf{w}^{(t+1)} \mapsfrom \mathbf{w}^{(t)} - \rho^{(t)}\nabla_{\mathbf{w}^{(t)}} - \gamma^{(t)}\mathbf{w}^{(t)}$$

Since by assumption $\mathbf{w}_1$ and w_2 are initialized near zero, $\mathbf{w}$ will initialize near zero as well. This implies that at every iteration t, $\mathbf{w}^{(t)}$ is a weighted combination of past gradients. There thus exist $\mu^{(t,\tau)} \in \mathbb{R}$ such that:

$$\mathbf{w}^{(t+1)} \mapsfrom \mathbf{w}^{(t)} - \rho^{(t)}\nabla_{\mathbf{w}^{(t)}} - \sum\nolimits_{\tau=1}^{t-1}\mu^{(t,\tau)}\nabla_{\mathbf{w}^{(\tau)}}$$

We conclude that the dynamics governing the underlying parameter $\mathbf{w}$ correspond to gradient descent with a momentum term, where both the learning rate ($\rho^{(t)}$) and momentum coefficients ($\mu^{(t,\tau)}$) are time-varying and adaptive.

4. Linear Neural Networks

Let $\mathcal{X} := \mathbb{R}^d$ be a space of objects (*e.g.* images or word embeddings) that we would like to infer something about, and let $\mathcal{Y} := \mathbb{R}^k$ be the space of possible inferences. Suppose we are given a training set $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^m \subset \mathcal{X} \times \mathcal{Y}$, along with a (point-wise) loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$. For example, $\mathbf{y}^{(i)}$ could hold continuous values with $l(\cdot)$ being the ℓ_2 loss: $l(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\left\|\hat{\mathbf{y}} - \mathbf{y}\right\|_2^2$; or it could hold one-hot vectors representing categories with $l(\cdot)$ being the softmax-cross-entropy loss: $l(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{r=1}^k y_r \log(e^{\hat{y}_r}/\sum_{r'=1}^k e^{\hat{y}_{r'}})$, where y_r and $\hat{y}_r$ stand for coordinate r of $\mathbf{y}$ and $\hat{\mathbf{y}}$ respectively. For a predictor ϕ, *i.e.* a mapping from $\mathcal{X}$ to $\mathcal{Y}$, the overall training loss is $L(\phi) := \frac{1}{m}\sum_{i=1}^m l(\phi(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$. If ϕ comes from some parametric family $\Phi := \{\phi_\theta : \mathcal{X} \to \mathcal{Y} | \theta \in \Theta\}$, we view the corresponding training loss as a function of the parameters, *i.e.* we consider $L^\Phi(\theta) := \frac{1}{m}\sum_{i=1}^m l(\phi_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$. For example, if the parametric family in question is the class of (directly parameterized) linear predictors:

$$\Phi^{lin} := \{\mathbf{x} \mapsto W\mathbf{x} | W \in \mathbb{R}^{k,d}\} \tag{1}$$

the respective training loss is a function from $\mathbb{R}^{k,d}$ to $\mathbb{R}_{\geq 0}$.

In our context, a depth-N ($N \geq 2$) linear neural network, with hidden widths $n_1, n_2, \ldots, n_{N-1} \in \mathbb{N}$, is the following parametric family of linear predictors: $\Phi^{n_1 \ldots n_{N-1}} := \{\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} | W_j \in \mathbb{R}^{n_j, n_{j-1}}, j{=}1...N\}$, where by definition $n_0 := d$ and $n_N := k$. As customary, we refer to each W_j, $j{=}1...N$, as the weight matrix of layer j. For simplicity of presentation, we hereinafter omit from our notation the hidden widths $n_1...n_{N-1}$, and simply write Φ^N instead of $\Phi^{n_1 \cdots n_{N-1}}$ ($n_1 \ldots n_{N-1}$ will be specified explicitly if not clear by context). That is, we denote:

$$\begin{aligned}&\Phi^N := \\ &\{\mathbf{x} \mapsto W_N W_{N-1} \cdots W_1 \mathbf{x} |\ W_j \in \mathbb{R}^{n_j, n_{j-1}},\ j{=}1...N\}\end{aligned} \tag{2}$$

For completeness, we regard a depth-1 network as the family of directly parameterized linear predictors, *i.e.* we set $\Phi^1 := \Phi^{lin}$ (see Equation 1).

The training loss that corresponds to a depth-N linear network – $L^{\Phi^N}(W_1, ..., W_N)$, is a function from $\mathbb{R}^{n_1, n_0} \times \cdots \times \mathbb{R}^{n_N, n_{N-1}}$ to $\mathbb{R}_{\geq 0}$. For brevity, we will denote this function by $L^N(\cdot)$. Our focus lies on the behavior of gradient descent when minimizing $L^N(\cdot)$. More specifically, we are interested in the dependence of this behavior on N, and in particular, in the possibility of increasing N leading to acceleration. Notice that for any $N \geq 2$ we have:

$$L^N(W_1, ..., W_N) = L^1(W_N W_{N-1} \cdots W_1) \tag{3}$$

and so the sole difference between the training loss of a depth-N network and that of a depth-1 network (classic linear model) lies in the replacement of a matrix parameter by a product of N matrices. This implies that if increasing N can indeed accelerate convergence, it is not an outcome of any phenomenon other than favorable properties of depth-induced overparameterization for optimization.

5. Implicit Dynamics of Gradient Descent

In this section we present a new result for linear neural networks, tying the dynamics of gradient descent on $L^N(\cdot)$ – the training loss corresponding to a depth-N network, to those on $L^1(\cdot)$ – training loss of a depth-1 network (classic linear model). Specifically, we show that gradient descent on $L^N(\cdot)$, a complicated and seemingly pointless overparameterization, can be directly rewritten as a particular preconditioning scheme over gradient descent on $L^1(\cdot)$.

When applied to $L^N(\cdot)$, gradient descent takes on the following form:

$$W_j^{(t+1)} \mapsfrom (1-\eta\lambda)W_j^{(t)} - \eta\frac{\partial L^N}{\partial W_j}(W_1^{(t)},\ldots,W_N^{(t)}) \quad , \; j=1\ldots N \tag{4}$$

$\eta > 0$ here is a learning rate, and $\lambda \geq 0$ is an optional weight decay coefficient. For simplicity, we regard both η and λ as fixed (no dependence on t). Define the underlying *end-to-end weight matrix*:

$$W_e := W_N W_{N-1} \cdots W_1 \tag{5}$$

Given that $L^N(W_1,\ldots,W_N) = L^1(W_e)$ (Equation 3), we view W_e as an optimized weight matrix for $L^1(\cdot)$, whose dynamics are governed by Equation 4. Our interest then boils down to the study of these dynamics for different choices of N. For $N = 1$ they are (trivially) equivalent to standard gradient descent over $L^1(\cdot)$. We will characterize the dynamics for $N \geq 2$.

To be able to derive, in our general setting, an explicit update rule for the end-to-end weight matrix W_e (Equation 5), we introduce an assumption by which the learning rate is small, *i.e.* $\eta^2 \approx 0$. Formally, this amounts to translating Equation 4 to the following set of differential equations:

$$\dot{W}_j(t) = -\eta\lambda W_j(t) - \eta\frac{\partial L^N}{\partial W_j}(W_1(t),\ldots,W_N(t)) \quad , \; j=1\ldots N \tag{6}$$

where t is now a continuous time index, and $\dot{W}_j(t)$ stands for the derivative of W_j with respect to time. The use of differential equations, for both theoretical analysis and algorithm design, has a long and rich history in optimization research (see Helmke & Moore (2012) for an overview). When step sizes (learning rates) are taken to be small, trajectories of discrete optimization algorithms converge to smooth curves modeled by continuous-time differential equations, paving way to the well-established theory of the latter (*cf.* Boyce et al. (1969)). This approach has led to numerous interesting findings, including recent results in the context of acceleration methods (*e.g.* Su et al. (2014); Wibisono et al. (2016)).

With the continuous formulation in place, we turn to express the dynamics of the end-to-end matrix W_e:

Theorem 1. *Assume the weight matrices $W_1 \ldots W_N$ follow the dynamics of continuous gradient descent (Equation 6). Assume also that their initial values (time t_0) satisfy, for $j = 1 \ldots N-1$:*

$$W_{j+1}^\top(t_0)W_{j+1}(t_0) = W_j(t_0)W_j^\top(t_0) \tag{7}$$

Then, the end-to-end weight matrix W_e (Equation 5) is governed by the following differential equation:

$$\begin{aligned}\dot{W}_e(t) = &-\eta\lambda N \cdot W_e(t) \\ &-\eta \sum\nolimits_{j=1}^{N} \left[W_e(t)W_e^\top(t)\right]^{\frac{j-1}{N}} \cdot \\ &\qquad \tfrac{dL^1}{dW}(W_e(t)) \cdot \left[W_e^\top(t)W_e(t)\right]^{\frac{N-j}{N}}\end{aligned} \tag{8}$$

where $[\cdot]^{\frac{j-1}{N}}$ and $[\cdot]^{\frac{N-j}{N}}$, $j = 1 \ldots N$, are fractional power operators defined over positive semidefinite matrices.

Proof. (sketch – full details in Appendix A.1) If $\lambda = 0$ (no weight decay) then one can easily show that $W_{j+1}^\top(t)\dot{W}_{j+1}(t) = \dot{W}_j(t)W_j^\top(t)$ throughout optimization. Taking the transpose of this equation and adding to itself, followed by integration over time, imply that the difference between $W_{j+1}^\top(t)W_{j+1}(t)$ and $W_j(t)W_j^\top(t)$ is constant. This difference is zero at initialization (Equation 7), thus will remain zero throughout, *i.e.*:

$$W_{j+1}^\top(t)W_{j+1}(t) = W_j(t)W_j^\top(t) \quad , \; \forall t \geq t_0 \tag{9}$$

A slightly more delicate treatment shows that this is true even if $\lambda > 0$, *i.e.* with weight decay included.

Equation 9 implies alignment of the (left and right) singular spaces of $W_j(t)$ and $W_{j+1}(t)$, simplifying the product $W_{j+1}(t)W_j(t)$. Successive application of this simplification allows a clean computation for the product of all layers (that is, W_e), leading to the explicit form presented in theorem statement (Equation 8). □

Translating the continuous dynamics of Equation 8 back to discrete time, we obtain the sought-after update rule for the end-to-end weight matrix:

$$\begin{aligned}W_e^{(t+1)} \mapsfrom &(1-\eta\lambda N)W_e^{(t)} \\ &-\eta \sum\nolimits_{j=1}^{N} \left[W_e^{(t)}(W_e^{(t)})^\top\right]^{\frac{j-1}{N}} \cdot \\ &\qquad \tfrac{dL^1}{dW}(W_e^{(t)}) \cdot \left[(W_e^{(t)})^\top W_e^{(t)}\right]^{\frac{N-j}{N}}\end{aligned} \tag{10}$$

This update rule relies on two assumptions: first, that the learning rate η is small enough for discrete updates to approximate continuous ones; and second, that weights are initialized on par with Equation 7, which will approximately be the case if initialization values are close enough to zero. It is customary in deep learning for both learning rate and weight initializations to be small, but nonetheless above assumptions are only met to a certain extent. We support their applicability by showing empirically (Section 8) that the end-to-end update rule (Equation 10) indeed provides an accurate description for the dynamics of W_e.

A close look at Equation 10 reveals that the dynamics of the end-to-end weight matrix W_e are similar to gradient descent over $L^1(\cdot)$ – training loss corresponding to a depth-1 network (classic linear model). The only difference (besides the scaling by N of the weight decay coefficient λ) is that the gradient $\frac{dL^1}{dW}(W_e)$ is subject to a transformation before being used. Namely, for $j = 1 \ldots N$, it is multiplied from the left by $[W_e W_e^\top]^{\frac{j-1}{N}}$ and from the right by $[W_e^\top W_e]^{\frac{N-j}{N}}$, followed by summation over j. Clearly, when $N = 1$ (depth-1 network) this transformation reduces to identity, and as expected, W_e precisely adheres to gradient descent over $L^1(\cdot)$. When $N \geq 2$ the dynamics of W_e are less interpretable. We arrange it as a vector to gain more insight:

Claim 1. *For an arbitrary matrix A, denote by $vec(A)$ its arrangement as a vector in column-first order. Then, the end-to-end update rule in Equation 10 can be written as:*

$$\begin{aligned} vec(W_e^{(t+1)}) \mapsfrom (1 - \eta\lambda N) \cdot vec(W_e^{(t)}) \\ -\eta \cdot P_{W_e^{(t)}} vec\left(\tfrac{dL^1}{dW}(W_e^{(t)})\right) \end{aligned} \tag{11}$$

where $P_{W_e^{(t)}}$ is a positive semidefinite preconditioning matrix that depends on $W_e^{(t)}$. Namely, if we denote the singular values of $W_e^{(t)} \in \mathbb{R}^{k,d}$ by $\sigma_1 \ldots \sigma_{\max\{k,d\}} \in \mathbb{R}_{\geq 0}$ (by definition $\sigma_r = 0$ if $r > \min\{k, d\}$), and corresponding left and right singular vectors by $\mathbf{u}_1 \ldots \mathbf{u}_k \in \mathbb{R}^k$ and $\mathbf{v}_1 \ldots \mathbf{v}_d \in \mathbb{R}^d$ respectively, the eigenvectors of $P_{W_e^{(t)}}$ are:

$$vec(\mathbf{u}_r \mathbf{v}_{r'}^\top) \quad , r = 1 \ldots k \, , \; r' = 1 \ldots d$$

with corresponding eigenvalues:

$$\sum\nolimits_{j=1}^{N} \sigma_r^{2\frac{N-j}{N}} \sigma_{r'}^{2\frac{j-1}{N}} \quad , r = 1 \ldots k \, , \; r' = 1 \ldots d$$

Proof. The result readily follows from the properties of the Kronecker product – see Appendix A.2 for details. □

Claim 1 implies that in the end-to-end update rule of Equation 10, the transformation applied to the gradient $\frac{dL^1}{dW}(W_e)$ is essentially a preconditioning, whose eigendirections and eigenvalues depend on the singular value decomposition of W_e. The eigendirections are the rank-1 matrices $\mathbf{u}_r \mathbf{v}_{r'}^\top$, where $\mathbf{u}_r$ and $\mathbf{v}_{r'}$ are left and right (respectively) singular vectors of W_e. The eigenvalue of $\mathbf{u}_r \mathbf{v}_{r'}^\top$ is $\sum_{j=1}^{N} \sigma_r^{2(N-j)/N} \sigma_{r'}^{2(j-1)/N}$, where σ_r and $\sigma_{r'}$ are the singular values of W_e corresponding to $\mathbf{u}_r$ and $\mathbf{v}_{r'}$ (respectively). When $N \geq 2$, an increase in σ_r or $\sigma_{r'}$ leads to an increase in the eigenvalue corresponding to the eigendirection $\mathbf{u}_r \mathbf{v}_{r'}^\top$. Qualitatively, this implies that the preconditioning favors directions that correspond to singular vectors whose presence in W_e is stronger. We conclude that the effect of overparameterization, *i.e.* of replacing a classic linear model (depth-1 network) by a depth-N linear network, boils down to modifying gradient descent by promoting movement along directions that fall in line with the current location in parameter space. A-priori, such a preference may seem peculiar – why should an optimization algorithm be sensitive to its location in parameter space? Indeed, we generally expect sensible algorithms to be translation invariant, *i.e.* be oblivious to parameter value. However, if one takes into account the common practice in deep learning of initializing weights near zero, the location in parameter space can also be regarded as the overall movement made by the algorithm. We thus interpret our findings as indicating that overparameterization promotes movement along directions already taken by the optimization, and therefore can be seen as a form of acceleration. This intuitive interpretation will become more concrete in the subsection that follows.

A final point to make, is that the end-to-end update rule (Equation 10 or 11), which obviously depends on N – number of layers in the deep linear network, does *not* depend on the hidden widths $n_1 \ldots n_{N-1}$ (see Section 4). This implies that from an optimization perspective, overparameterizing using wide or narrow networks has the same effect – it is only the depth that matters. Consequently, the acceleration of overparameterization can be attained at a minimal computational price, as we demonstrate empirically in Section 8.

5.1. Single Output Case

To facilitate a straightforward presentation of our findings, we hereinafter focus on the special case where the optimized models have a single output, *i.e.* where $k = 1$. This corresponds, for example, to a binary (two-class) classification problem, or to the prediction of a numeric scalar property (regression). It admits a particularly simple form for the end-to-end update rule of Equation 10:

Claim 2. *Assume $k = 1$,* i.e. $W_e \in \mathbb{R}^{1,d}$. *Then, the end-to-end update rule in Equation 10 can be written as follows:*

$$\begin{aligned} W_e^{(t+1)} \mapsfrom (1 - \eta\lambda N) \cdot W_e^{(t)} \\ -\eta \|W_e^{(t)}\|_2^{2-\frac{2}{N}} \cdot \Big(\tfrac{dL^1}{dW}(W_e^{(t)}) + \\ (N-1) \cdot Pr_{W_e^{(t)}}\{\tfrac{dL^1}{dW}(W_e^{(t)})\}\Big) \end{aligned} \tag{12}$$

where $\|\cdot\|_2^{2-\frac{2}{N}}$ stands for Euclidean norm raised to the power of $2 - \frac{2}{N}$, and $Pr_W\{\cdot\}$, $W \in \mathbb{R}^{1,d}$, is defined to be the projection operator onto the direction of W:

$$\begin{aligned} Pr_W : \mathbb{R}^{1,d} \to \mathbb{R}^{1,d} \\ Pr_W\{V\} := \begin{cases} \frac{W}{\|W\|_2} V^\top \cdot \frac{W}{\|W\|_2} & , W \neq 0 \\ 0 & , W = 0 \end{cases} \end{aligned} \tag{13}$$

Proof. The result follows from the definition of a fractional power operator over matrices – see Appendix A.3. □

Claim 2 implies that in the single output case, the effect of overparameterization (replacing classic linear model by depth-N linear network) on gradient descent is twofold: first, it leads to an *adaptive learning rate* schedule, by introducing the multiplicative factor $\|W_e\|_2^{2-2/N}$; and second, it amplifies (by N) the projection of the gradient on the direction of W_e. Recall that we view W_e not only as the optimized parameter, but also as the overall movement made in optimization (initialization is assumed to be near zero). Accordingly, the adaptive learning rate schedule can be seen as gaining confidence (increasing step sizes) when optimization moves farther away from initialization, and the gradient projection amplification can be thought of as a certain type of *momentum* that favors movement along the azimuth taken so far. These effects bear potential to accelerate convergence, as we illustrate qualitatively in Section 7, and demonstrate empirically in Section 8.

6. Overparametrization Effects Cannot Be Attained via Regularization

Adding a regularizer to the objective is a standard approach for improving optimization (though lately the term regularization is typically associated with generalization). For example, AdaGrad was originally invented to compete with the best regularizer from a particular family. The next theorem shows (for single output case) that the effects of overparameterization cannot be attained by adding a regularization term to the original training loss, or via any similar modification. This is not obvious a-priori, as unlike many acceleration methods that explicitly maintain memory of past gradients, updates under overparametrization are by definition the gradients of *something*. The assumptions in the theorem are minimal and also necessary, as one must rule-out the trivial counter-example of a constant training loss.

Theorem 2. *Assume* $\frac{dL^1}{dW}$ *does not vanish at* $W = 0$*, and is continuous on some neighborhood around this point. For a given* $N \in \mathbb{N}$*,* $N > 2$*,*[1] *define:*

$$F(W) := \|W\|_2^{2-\frac{2}{N}} \cdot \left(\frac{dL^1}{dW}(W) + (N{-}1) \cdot Pr_W\left\{\frac{dL^1}{dW}(W)\right\}\right) \tag{14}$$

where $Pr_W\{\cdot\}$ *is the projection given in Equation 13. Then, there exists no function (of* W*) whose gradient field is* $F(\cdot)$*.*

Proof. (sketch – full details in Appendix A.4) The proof uses elementary differential geometry (Buck, 2003): curves, arc length and the fundamental theorem for line integrals, which states that the integral of ∇g for any differentiable function g amounts to 0 along every closed curve.

Overparametrization changes gradient descent's behavior: instead of following the original gradient $\frac{dL^1}{dW}$, it follows some other direction $F(\cdot)$ (see Equations 12 and 14) that is a *function* of the original gradient as well as the current point W. We think of this change as a transformation that maps one *vector field* $\phi(\cdot)$ to another – $F_\phi(\cdot)$:

$$F_\phi(W) = \begin{cases} \|W\|^{2-\frac{2}{N}}\left(\phi(W){+}(N{-}1)\left\langle\phi(W), \frac{W}{\|W\|}\right\rangle\frac{W}{\|W\|}\right) & , W{\neq}0 \\ 0 & , W{=}0 \end{cases}$$

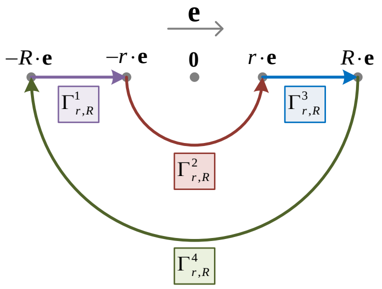

Figure 1. Curve $\Gamma_{r,R}$ over which line integral is non-zero.

Notice that for $\phi = \frac{dL^1}{dW}$, we get exactly the vector field $F(\cdot)$ defined in theorem statement.

We note simple properties of the mapping $\phi \mapsto F_\phi$. First, it is linear, since for any vector fields ϕ_1, ϕ_2 and scalar c: $F_{\phi_1+\phi_2}{=}F_{\phi_1}{+}F_{\phi_2}$ and $F_{c\cdot\phi_1}{=}c{\cdot}F_{\phi_1}$. Second, because of the linearity of line integrals, for any curve Γ, the functional $\phi \mapsto \int_\Gamma F_\phi$, a mapping of vector fields to scalars, is linear.

We show that $F(\cdot)$ contradicts the fundamental theorem for line integrals. To do so, we construct a closed curve $\Gamma{=}\Gamma_{r,R}$ for which the linear functional $\phi \mapsto \oint_\Gamma F_\phi$ does not vanish at $\phi{=}\frac{dL^1}{dW}$. Let $\mathbf{e} := \frac{dL^1}{dW}(W{=}0)/\|\frac{dL^1}{dW}(W{=}0)\|$, which is well-defined since by assumption $\frac{dL^1}{dW}(W{=}0){\neq}\ 0$. For $r < R$ we define (see Figure 1):

$$\Gamma_{r,R} := \Gamma^1_{r,R} \ \rightarrow\ \Gamma^2_{r,R} \ \rightarrow\ \Gamma^3_{r,R} \ \rightarrow\ \Gamma^4_{r,R}$$

where:

- $\Gamma^1_{r,R}$ is the line segment from $-R \cdot \mathbf{e}$ to $-r \cdot \mathbf{e}$.
- $\Gamma^2_{r,R}$ is a spherical curve from $-r \cdot \mathbf{e}$ to $r \cdot \mathbf{e}$.
- $\Gamma^3_{r,R}$ is the line segment from $r \cdot \mathbf{e}$ to $R \cdot \mathbf{e}$.
- $\Gamma^4_{r,R}$ is a spherical curve from $R \cdot \mathbf{e}$ to $-R \cdot \mathbf{e}$.

With the definition of $\Gamma_{r,R}$ in place, we decompose $\frac{dL^1}{dW}$ into a constant vector field $\kappa \equiv \frac{dL^1}{dW}(W{=}0)$ plus a residual ξ. We explicitly compute the line integrals along $\Gamma^1_{r,R} \ldots \Gamma^4_{r,R}$ for F_κ, and derive bounds for F_ξ. This, along with the linearity of the functional $\phi \mapsto \int_\Gamma F_\phi$, provides a lower bound on the line integral of $F(\cdot)$ over $\Gamma_{r,R}$. We show the lower bound is positive as $r, R \to 0$, thus $F(\cdot)$ indeed contradicts the fundamental theorem for line integrals. □

7. Illustration of Acceleration

To this end, we showed that overparameterization (use of depth-N linear network in place of classic linear model) induces on gradient descent a particular preconditioning scheme (Equation 10 in general and 12 in the single output

[1] For the result to hold with $N = 2$, additional assumptions on $L^1(\cdot)$ are required; otherwise any non-zero linear function $L^1(W) = WU^\top$ serves as a counter-example – it leads to a vector field $F(\cdot)$ that is the gradient of $W \mapsto \|W\|_2 \cdot WU^\top$.

case), which can be interpreted as introducing some forms of momentum and adaptive learning rate. We now illustrate qualitatively, on a very simple hypothetical learning problem, the potential of these to accelerate optimization.

Consider the task of linear regression, assigning to vectors in $\mathbb{R}^2$ labels in $\mathbb{R}$. Suppose that our training set consists of two points in $\mathbb{R}^2 \times \mathbb{R}$: $([1,0]^\top, y_1)$ and $([0,1]^\top, y_2)$. Assume also that the loss function of interest is ℓ_p, $p \in 2\mathbb{N}$: $\ell_p(\hat{y}, y) = \frac{1}{p}(\hat{y} - y)^p$. Denoting the learned parameter by $\mathbf{w} = [w_1, w_2]^\top$, the overall training loss can be written as:[2]

$$L(w_1, w_2) = \tfrac{1}{p}(w_1 - y_1)^p + \tfrac{1}{p}(w_2 - y_2)^p$$

With fixed learning rate $\eta > 0$ (weight decay omitted for simplicity), gradient descent over $L(\cdot)$ gives:

$$w_i^{(t+1)} \hookleftarrow w_i^{(t)} - \eta(w_i^{(t)} - y_i)^{p-1} \quad , \, i = 1, 2$$

Changing variables per $\Delta_i = w_i - y_i$, we have:

$$\Delta_i^{(t+1)} \hookleftarrow \Delta_i^{(t)} \big(1 - \eta(\Delta_i^{(t)})^{p-2}\big) \quad , \, i = 1, 2 \qquad (15)$$

Assuming the original weights w_1 and w_2 are initialized near zero, Δ_1 and Δ_2 start off at $-y_1$ and $-y_2$ respectively, and will eventually reach the optimum $\Delta_1^* = \Delta_2^* = 0$ if the learning rate is small enough to prevent divergence:

$$\eta < \tfrac{2}{y_i^{p-2}} \quad , \, i = 1, 2$$

Suppose now that the problem is ill-conditioned, in the sense that $y_1 \gg y_2$. If $p = 2$ this has no effect on the bound for η.[3] If $p > 2$ the learning rate is determined by y_1, leading Δ_2 to converge very slowly. In a sense, Δ_2 will suffer from the fact that there is no "communication" between the coordinates (this will actually be the case not just with gradient descent, but with most algorithms typically used in large-scale settings – AdaGrad, Adam, *etc.*).

Now consider the scenario where we optimize $L(\cdot)$ via overparameterization, *i.e.* with the update rule in Equation 12 (single output). In this case the coordinates are coupled, and as Δ_1 gets small (w_1 gets close to y_1), the learning rate is effectively scaled by $y_1^{2-\frac{2}{N}}$ (in addition to a scaling by N in coordinate 1 only), allowing (if $y_1 > 1$) faster convergence of Δ_2. We thus have the luxury of temporarily slowing down Δ_2 to ensure that Δ_1 does not diverge, with the latter speeding up the former as it reaches safe grounds. In Appendix B we consider a special case and formalize this intuition, deriving a concrete bound for the acceleration of overparameterization.

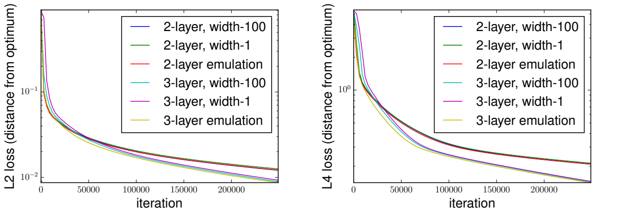

Figure 2. (to be viewed in color) Gradient descent optimization of deep linear networks (depths $2, 3$) *vs.* the analytically-derived equivalent preconditioning schemes (over single layer model; Equation 12). Both plots show training objective (left – ℓ_2 loss; right – ℓ_4 loss) per iteration, on a numeric regression dataset from UCI Machine Learning Repository (details in text). Notice the emulation of preconditioning schemes. Notice also the negligible effect of network width – for a given depth, setting size of hidden layers to 1 (scalars) or 100 yielded similar convergence (on par with our analysis).

8. Experiments

Our analysis (Section 5) suggests that overparameterization – replacement of a classic linear model by a deep linear network, induces on gradient descent a certain preconditioning scheme. We qualitatively argued (Section 7) that in some cases, this preconditioning may accelerate convergence. In this section we put these claims to the test, through a series of empirical evaluations based on TensorFlow toolbox (Abadi et al. (2016)). For conciseness, many of the details behind our implementation are deferred to Appendix C.

We begin by evaluating our analytically-derived preconditioning scheme – the end-to-end update rule in Equation 10. Our objective in this experiment is to ensure that our analysis, continuous in nature and based on a particular assumption on weight initialization (Equation 7), is indeed applicable to practical scenarios. We focus on the single output case, where the update-rule takes on a particularly simple (and efficiently implementable) form – Equation 12. The dataset chosen was UCI Machine Learning Repository's "Gas Sensor Array Drift at Different Concentrations" (Vergara et al., 2012; Rodriguez-Lujan et al., 2014). Specifically, we used the dataset's "Ethanol" problem – a scalar regression task with 2565 examples, each comprising 128 features (one of the largest numeric regression tasks in the repository). As training objectives, we tried both ℓ_2 and ℓ_4 losses. Figure 2 shows convergence (training objective per iteration) of gradient descent optimizing depth-2 and depth-3 linear networks, against optimization of a single layer model using the respective preconditioning schemes (Equation 12 with $N = 2, 3$). As can be seen, the preconditioning schemes reliably emulate deep network optimization, suggesting that, at least in some cases, our analysis indeed captures practical dynamics.

Alongside the validity of the end-to-end update rule, Figure 2 also demonstrates the negligible effect of network width on convergence, in accordance with our analysis (see Section 5). Specifically, it shows that in the evaluated setting, hidden layers of size 1 (scalars) suffice in order for the essence of overparameterization to fully emerge. Unless oth-

[2] We omit the averaging constant $\frac{1}{2}$ for conciseness.

[3] Optimal learning rate for gradient descent on quadratic objective does not depend on current parameter value (*cf.* Goh (2017)).

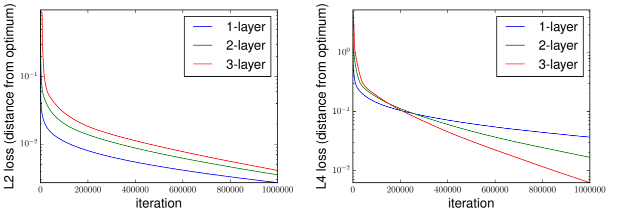

Figure 3. (to be viewed in color) Gradient descent optimization of single layer model *vs.* linear networks of depth 2 and 3. Setup is identical to that of Figure 2, except that here learning rates were chosen via grid search, individually per model (see Appendix C). Notice that with ℓ_2 loss, depth (slightly) hinders optimization, whereas with ℓ_4 loss it leads to significant acceleration (on par with our qualitative analysis in Section 7).

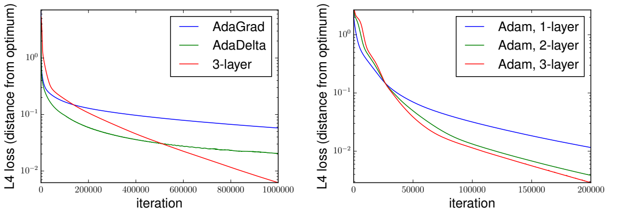

Figure 4. (to be viewed in color) **Left:** Gradient descent optimization of depth-3 linear network *vs.* AdaGrad and AdaDelta over single layer model. Setup is identical to that of Figure 3-right. Notice that the implicit acceleration of overparameterization outperforms both AdaGrad and AdaDelta (former is actually slower than plain gradient descent). **Right:** Adam optimization of single layer model *vs.* Adam over linear networks of depth 2 and 3. Same setup, but with learning rates set per Adam's default in TensorFlow. Notice that depth improves speed, suggesting that the acceleration of overparameterization may be somewhat orthogonal to explicit acceleration methods.

erwise indicated, all results reported hereinafter are based on this configuration, *i.e.* on scalar hidden layers. The computational toll associated with overparameterization will thus be virtually non-existent.

As a final observation on Figure 2, notice that it exhibits faster convergence with a deeper network. This however does not serve as evidence in favor of acceleration by depth, as we did not set learning rates optimally per model (simply used the common choice of 10^{-3}). To conduct a fair comparison between the networks, and more importantly, between them and a classic single layer model, multiple learning rates were tried, and the one giving fastest convergence was taken on a per-model basis. Figure 3 shows the results of this experiment. As can be seen, convergence of deeper networks is (slightly) slower in the case of ℓ_2 loss. This falls in line with the findings of Saxe et al. (2013). In stark contrast, and on par with our qualitative analysis in Section 7, is the fact that with ℓ_4 loss adding depth significantly accelerated convergence. To the best of our knowledge, this provides first empirical evidence to the fact that depth, even without any gain in expressiveness, and despite introducing non-convexity to a formerly convex problem, can lead to favorable optimization.

In light of the speedup observed with ℓ_4 loss, it is natural to ask how the implicit acceleration of depth compares against explicit methods for acceleration and adaptive learning. Figure 4-left shows convergence of a depth-3 network (optimized with gradient descent) against that of a single layer model optimized with AdaGrad (Duchi et al., 2011) and AdaDelta (Zeiler, 2012). The displayed curves correspond to optimal learning rates, chosen individually via grid search. Quite surprisingly, we find that in this specific setting, overparameterizing, thereby turning a convex problem non-convex, is a more effective optimization strategy than carefully designed algorithms tailored for convex problems. We note that this was not observed with all algorithms – for example Adam (Kingma & Ba, 2014) was considerably faster than overparameterization. However, when introducing overparameterization simultaneously with Adam (a setting we did not theoretically analyze), further acceleration is attained – see Figure 4-right. This suggests that at least in some cases, not only plain gradient descent benefits from depth, but also more elaborate algorithms commonly employed in state of the art applications.

An immediate question arises at this point. If depth indeed accelerates convergence, why not add as many layers as one can computationally afford? The reason, which is actually apparent in our analysis, is the so-called *vanishing gradient problem*. When training a very deep network (large N), while initializing weights to be small, the end-to-end matrix W_e (Equation 5) is extremely close to zero, severely attenuating gradients in the preconditioning scheme (Equation 10). A possible approach for alleviating this issue is to initialize weights to be larger, yet small enough such that the end-to-end matrix does not "explode". The choice of identity (or near identity) initialization leads to what is known as *linear residual networks* (Hardt & Ma, 2016), akin to the successful residual networks architecture (He et al., 2015) commonly employed in deep learning. Notice that identity initialization satisfies the condition in Equation 7, rendering the end-to-end update rule (Equation 10) applicable. Figure 5-left shows convergence, under gradient descent, of a single layer model against deeper networks than those evaluated before – depths 4 and 8. As can be seen, with standard, near-zero initialization, the depth-4 network starts making visible progress only after about $65K$ iterations, whereas the depth-8 network seems stuck even after $100K$ iterations. In contrast, under identity initialization, both networks immediately make progress, and again depth serves as an implicit accelerator.

As a final sanity test, we evaluate the effect of overparameterization on optimization in a non-idealized (yet simple) deep learning setting. Specifically, we experiment with the convolutional network tutorial for MNIST built into Ten-

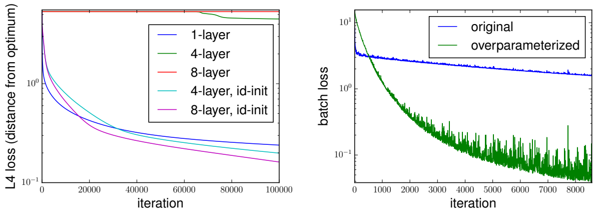

Figure 5. (to be viewed in color) **Left:** Gradient descent optimization of single layer model *vs.* linear networks deeper than before (depths $4, 8$). For deep networks, both near-zero and near-identity initializations were evaluated. Setup identical to that of Figure 3-right. Notice that deep networks suffer from vanishing gradients under near-zero initialization, while near-identity ("residual") initialization eliminates the problem. **Right:** Stochastic gradient descent optimization in TensorFlow's convolutional network tutorial for MNIST. Plot shows batch loss per iteration, in original setting *vs.* overparameterized one (depth-2 linear networks in place of dense layers).

sorFlow,[4] which includes convolution, pooling and dense layers, ReLU non-linearities, stochastic gradient descent with momentum, and dropout (Srivastava et al., 2014). We introduced overparameterization by simply placing two matrices in succession instead of the matrix in each dense layer. Here, as opposed to previous experiments, widths of the newly formed hidden layers were not set to 1, but rather to the minimal values that do not deteriorate expressiveness (see Appendix C). Overall, with an addition of roughly 15% in number of parameters, optimization has accelerated considerably – see Figure 5-right. The displayed results were obtained with the hyperparameter settings hardcoded into the tutorial. We have tried alternative settings (varying learning rates and standard deviations of initializations – see Appendix C), and in all cases observed an outcome similar to that in Figure 5-right – overparameterization led to significant speedup. Nevertheless, as reported above for linear networks, it is likely that for non-linear networks the effect of depth on optimization is mixed – some settings accelerate by it, while others do not. Comprehensive characterization of the cases in which depth accelerates optimization warrants much further study. We hope our work will spur interest in this avenue of research.

9. Conclusion

Through theory and experiments, we demonstrated that overparameterizing a neural network by increasing its depth can accelerate optimization, even on very simple problems.

Our analysis of linear neural networks, the subject of various recent studies, yielded a new result: for these models, overparameterization by depth can be understood as a preconditioning scheme with a closed form description (Theorem 1 and the claims thereafter). The preconditioning may be interpreted as a combination between certain forms of adaptive learning rate and momentum. Given that it depends on network depth but not on width, acceleration by overparameterization can be attained at a minimal computational price, as we demonstrate empirically in Section 8.

Clearly, complete theoretical analysis for non-linear networks will be challenging. Empirically however, we showed that the trivial idea of replacing an internal weight matrix by a product of two can significantly accelerate optimization, with absolutely no effect on expressiveness (Figure 5-right).

The fact that gradient descent over classic convex problems such as linear regression with ℓ_p loss, $p > 2$, can accelerate from transitioning to a non-convex overparameterized objective, does not coincide with conventional wisdom, and provides food for thought. Can this effect be rigorously quantified, similarly to analyses of explicit acceleration methods such as momentum or adaptive regularization (AdaGrad)?

Acknowledgments

Sanjeev Arora's work is supported by NSF, ONR, Simons Foundation, Schmidt Foundation, Mozilla Research, Amazon Research, DARPA and SRC. Elad Hazan's work is supported by NSF grant 1523815 and Google Brain. Nadav Cohen is a member of the Zuckerman Israeli Postdoctoral Scholars Program, and is supported by Eric and Wendy Schmidt.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima faster than gradient descent. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1195–1199. ACM, 2017.

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. Understanding deep neural networks with rectified linear units. *International Conference on Learning Representations (ICLR)*, 2018.

Baldi, P. and Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Boyce, W. E., DiPrima, R. C., and Haines, C. W. *Elementary differential equations and boundary value problems*, volume 9. Wiley New York, 1969.

Buck, R. C. *Advanced calculus*. Waveland Press, 2003.

Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. Accelerated methods for non-convex optimization. *arXiv preprint arXiv:1611.00756*, 2016.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pp. 192–204, 2015.

[4] https://github.com/tensorflow/models/tree/master/tutorials/image/mnist

Cohen, N., Sharir, O., Levine, Y., Tamari, R., Yakira, D., and Shashua, A. Analysis and design of convolutional networks via hierarchical tensor decompositions. *arXiv preprint arXiv:1705.02302*, 2017.

Daniely, A. Depth separation for neural networks. *arXiv preprint arXiv:1702.08489*, 2017.

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.

Eldan, R. and Shamir, O. The power of depth for feedforward neural networks. *arXiv preprint arXiv:1512.03965*, 2015.

Fukumizu, K. Effect of batch learning in multilayer neural networks. *Gen*, 1(04):1E–03, 1998.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pp. 797–842, 2015.

Goh, G. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Haeffele, B. D. and Vidal, R. Global Optimality in Tensor Factorization, Deep Learning, and Beyond. *CoRR abs/1202.2745*, cs.NA, 2015.

Hardt, M. and Ma, T. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

Hazan, E., Agarwal, A., and Kale, S. Logarithmic regret algorithms for online convex optimization. *Mach. Learn.*, 69(2-3):169–192, December 2007. ISSN 0885-6125.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Helmke, U. and Moore, J. B. *Optimization and dynamical systems*. Springer Science & Business Media, 2012.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456, 2015.

Janzamin, M., Sedghi, H., and Anandkumar, A. Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods. *CoRR abs/1506.08473*, 2015.

Jones, E., Oliphant, T., Peterson, P., et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed ¡today¿].

Kawaguchi, K. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pp. 586–594, 2016.

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

Lee, H., Ge, R., Risteski, A., Ma, T., and Arora, S. On the ability of neural nets to express distributions. *arXiv preprint arXiv:1702.07028*, 2017.

Livni, R., Shalev-Shwartz, S., and Shamir, O. On the computational efficiency of training neural networks. *Advances in Neural Information Processing Systems*, 2014.

Nesterov, Y. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pp. 372–376, 1983.

Nocedal, J. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. On the expressive power of deep neural networks. *arXiv preprint arXiv:1606.05336*, 2016.

Rodriguez-Lujan, I., Fonollosa, J., Vergara, A., Homer, M., and Huerta, R. On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems*, 130:123–134, 2014.

Safran, I. and Shamir, O. On the quality of the initial basin in overspecified neural networks. In *International Conference on Machine Learning*, pp. 774–782, 2016.

Safran, I. and Shamir, O. Spurious local minima are common in two-layer relu neural networks. *arXiv preprint arXiv:1712.08968*, 2017.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Soudry, D. and Carmon, Y. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

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

Su, W., Boyd, S., and Candes, E. A differential equation for modeling nesterovs accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pp. 2510–2518, 2014.

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

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., and Huerta, R. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166:320–329, 2012.

Wibisono, A., Wilson, A. C., and Jordan, M. I. A variational perspective on accelerated methods in optimization. *Proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.

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

A. Deferred Proofs

A.1. Proof of Theorem 1

Before delving into the proof, we introduce notation that will admit a more compact presentation of formulae. For $1 \leq a \leq b \leq N$, we denote:

$$\begin{aligned}\prod\nolimits_{a}^{j=b} W_j &:= W_b W_{b-1} \cdots W_a \\ \prod\nolimits_{j=a}^{b} W_j^\top &:= W_a^\top W_{a+1}^\top \cdots W_b^\top\end{aligned}$$

where $W_1 \ldots W_N$ are the weight matrices of the depth-N linear network (Equation 2). If $a > b$, then by definition both $\prod_{a}^{j=b} W_j$ and $\prod_{j=a}^{b} W_j^\top$ are identity matrices, with size depending on context, *i.e.* on the dimensions of matrices they are multiplied against. Given any square matrices (possibly scalars) $A_1, A_2, \ldots, A_m$, we denote by $diag(A_1 \ldots A_m)$ a block-diagonal matrix holding them on its diagonal:

$$diag(A_1 \ldots A_m) = \begin{bmatrix} A_1 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & A_m & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

As illustrated above, $diag(A_1 \ldots A_m)$ may hold additional, zero-valued rows and columns beyond $A_1 \ldots A_m$. Conversely, it may also trim (omit) rows and columns, from its bottom and right ends respectively, so long as only zeros are being removed. The exact shape of $diag(A_1 \ldots A_m)$ is again determined by context, and so if B and C are matrices, the expression $B \cdot diag(A_1 \ldots A_m) \cdot C$ infers a number of rows equal to the number of columns in B, and a number of columns equal to the number of rows in C.

Turning to the actual proof, we disregard the trivial case $N = 1$, and begin by noticing that Equation 3, along with the definition of W_e (Equation 5), imply that for every $j = 1 \ldots N$:

$$\frac{\partial L^N}{\partial W_j}(W_1, \ldots, W_N) = \prod_{i=j+1}^{N} W_i^\top \cdot \frac{dL^1}{dW}(W_e) \cdot \prod_{i=1}^{j-1} W_i^\top$$

Plugging this into the differential equations of gradient descent (Equation 6), we get:

$$\begin{aligned}\dot{W}_j(t) = &-\eta\lambda W_j(t) \qquad (16)\\ &-\eta \prod_{i=j+1}^{N} W_i^\top(t) \cdot \frac{dL^1}{dW}(W_e(t)) \cdot \prod_{i=1}^{j-1} W_i^\top(t) \\ &\qquad , \; j = 1 \ldots N\end{aligned}$$

For $j = 1 \ldots N{-}1$, multiply the j'th equation by $W_j^\top(t)$ from the right, and the $j{+}1$'th equation by $W_{j+1}^\top(t)$ from the left. This yields:

$$\begin{aligned}W_{j+1}^\top(t)\dot{W}_{j+1}(t) + \eta\lambda \cdot W_{j+1}^\top(t) W_{j+1}(t) = \\ \dot{W}_j(t) W_j^\top(t) + \eta\lambda \cdot W_j(t) W_j^\top(t)\end{aligned}$$

Taking the transpose of these equations and adding to themselves, we obtain, for every $j = 1 \ldots N{-}1$:

$$\begin{aligned}W_{j+1}^\top(t)\dot{W}_{j+1}(t) + \dot{W}_{j+1}^\top(t) W_{j+1}(t) + \\ 2\eta\lambda \cdot W_{j+1}^\top(t) W_{j+1}(t) = \\ \dot{W}_j(t) W_j^\top(t) + W_j(t)\dot{W}_j^\top(t) + \\ 2\eta\lambda \cdot W_j(t) W_j^\top(t) \qquad (17)\end{aligned}$$

Denote for $j = 1 \ldots N$:

$$C_j(t) := W_j(t) W_j^\top(t) \quad , \quad C_j'(t) := W_j^\top(t) W_j(t)$$

Equation 17 can now be written as:

$$\begin{aligned}\dot{C}_{j+1}'(t) + 2\eta\lambda \cdot C_{j+1}'(t) = \dot{C}_j(t) + 2\eta\lambda \cdot C_j(t) \\ , \; j = 1 \ldots N-1\end{aligned}$$

Turning to Lemma 1 below, while recalling our assumption for time t_0 (Equation 7):

$$C_{j+1}'(t_0) = C_j(t_0) \quad , \; j = 1 \ldots N-1$$

we conclude that, throughout the entire time-line:

$$C_{j+1}'(t) = C_j(t) \quad , \; j = 1 \ldots N-1$$

Recollecting the definitions of $C_j(t), C_j'(t)$, this means:

$$W_{j+1}^\top(t) W_{j+1}(t) = W_j(t) W_j^\top(t) \;\; , \; j = 1 \ldots N{-}1 \quad (18)$$

Regard t now as fixed, and for every $j = 1 \ldots N$, let:

$$W_j(t) = U_j \Sigma_j V_j^\top \qquad (19)$$

be a singular value decomposition. That is to say, U_j and V_j are orthogonal matrices, and Σ_j is a rectangular-diagonal matrix holding non-decreasing, non-negative singular values on its diagonal. Equation 18 implies that for $j = 1 \ldots N{-}1$:

$$V_{j+1} \Sigma_{j+1}^\top \Sigma_{j+1} V_{j+1}^\top = U_j \Sigma_j \Sigma_j^\top U_j^\top$$

For a given j, the two sides of the above equation are both orthogonal eigenvalue decompositions of the same matrix. The square-diagonal matrices $\Sigma_{j+1}^\top \Sigma_{j+1}$ and $\Sigma_j \Sigma_j^\top$ are thus the same, up to a possible permutation of diagonal elements (eigenvalues). However, since by definition Σ_{j+1} and Σ_j have non-increasing diagonals, it must hold that $\Sigma_{j+1}^\top \Sigma_{j+1} = \Sigma_j \Sigma_j^\top$. Let $\rho_1 > \rho_2 > \cdots > \rho_m \geq 0$ be the distinct eigenvalues, with corresponding multiplicities $d_1, d_2, \ldots, d_m \in \mathbb{N}$. We may write:

$$\Sigma_{j+1}^\top \Sigma_{j+1} = \Sigma_j \Sigma_j^\top = diag(\rho_1 I_{d_1}, \ldots, \rho_m I_{d_m}) \qquad (20)$$

where I_{d_r}, $1{\le}r{\le}m$, is the identity matrix of size $d_r \times d_r$. Moreover, there exist orthogonal matrices $O_{j,r} \in \mathbb{R}^{d_r,d_r}$, $1{\le}r{\le}m$, such that:

$$U_j = V_{j+1} \cdot diag(O_{j,1}, \ldots, O_{j,m})$$

$O_{j,r}$ here is simply a matrix changing between orthogonal bases in the eigenspace of ρ_r – it maps the basis comprising V_{j+1}-columns to that comprising U_j-columns. Recalling that both Σ_j and Σ_{j+1} are rectangular-diagonal, holding only non-negative values, Equation 20 implies that each of these matrices is equal to $diag(\sqrt{\rho_1}{\cdot}I_{d_1}, \ldots, \sqrt{\rho_m}{\cdot}I_{d_m})$. Note that the matrices generally do not have the same shape and thus, formally, are not equal to one another. Nonetheless, in line with our $diag$ notation (see beginning of this subsection), Σ_j and Σ_{j+1} may differ from each other only in trailing, zero-valued rows and columns. By an inductive argument, all the singular value matrices $\Sigma_1, \Sigma_2, \ldots, \Sigma_N$ (see Equation 19) are equal up to trailing zero rows and columns. The fact that $\rho_1 \ldots \rho_m$ do not include an index j in their notation is thus in order, and we may write, for every $j = 1 \ldots N{-}1$:

$$\begin{aligned} W_j(t) &= U_j \Sigma_j V_j^\top \\ &= V_{j+1} \cdot diag(O_{j,1}, \ldots, O_{j,m}) \cdot \\ &\qquad diag(\sqrt{\rho_1}{\cdot}I_{d_1}, \ldots, \sqrt{\rho_m}{\cdot}I_{d_m}) \cdot V_j^\top \end{aligned}$$

For the N'th weight matrix we have:

$$\begin{aligned} W_N(t) &= U_N \Sigma_N V_N^\top \\ &= U_N \cdot diag(\sqrt{\rho_1}{\cdot}I_{d_1}, \ldots, \sqrt{\rho_m}{\cdot}I_{d_m}) \cdot V_N^\top \end{aligned}$$

Concatenations of weight matrices thus simplify as follows:

$$\textstyle\prod_j^{i=N} W_i(t) \prod_{i=j}^N W_i^\top(t) = \tag{21}$$
$$U_N \cdot diag\Big((\rho_1)^{N-j+1}{\cdot}I_{d_1}, \ldots, (\rho_m)^{N-j+1}{\cdot}I_{d_m}\Big) \cdot U_N^\top$$

$$\textstyle\prod_{i=1}^j W_i^\top(t) \prod_1^{i=j} W_i(t) = \tag{22}$$
$$V_1 \cdot diag\Big((\rho_1)^j{\cdot}I_{d_1}, \ldots, (\rho_m)^j{\cdot}I_{d_m}\Big) \cdot V_1^\top$$
$$, \; j = 1 \ldots N$$

where we used the orthogonality of $O_{j,r}$, and the obvious fact that it commutes with I_{d_r}. Consider Equation 21 with $j = 1$ and Equation 22 with $j = N$, while recalling that by definition $W_e(t) = \prod_1^{i=N} W_j(t)$:

$$W_e(t)W_e^\top(t) = U_N{\cdot}diag\Big((\rho_1)^N I_{d_1}, \ldots, (\rho_m)^N I_{d_m}\Big){\cdot}U_N^\top$$
$$W_e^\top(t)W_e(t) = V_1{\cdot}diag\Big((\rho_1)^N I_{d_1}, \ldots, (\rho_m)^N I_{d_m}\Big){\cdot}V_1^\top$$

It follows that for every $j = 1 \ldots N$:

$$\prod_j^{i=N} W_i(t) \prod_{i=j}^N W_i^\top(t) = \left[W_e(t)W_e^\top(t)\right]^{\frac{N-j+1}{N}} \tag{23}$$

$$\prod_{i=1}^j W_i^\top(t) \prod_1^{i=j} W_i(t) = \left[W_e^\top(t)W_e(t)\right]^{\frac{j}{N}} \tag{24}$$

where $[\cdot]^{\frac{N-j+1}{N}}$ and $[\cdot]^{\frac{j}{N}}$ stand for fractional power operators defined over positive semidefinite matrices.

With Equations 23 and 24 in place, we are finally in a position to complete the proof. Returning to Equation 16, we multiply $\dot{W}_j(t)$ from the left by $\prod_{j+1}^{i=N} W_i(t)$ and from the right by $\prod_1^{i=j-1} W_i(t)$, followed by summation over $j = 1 \ldots N$. This gives:

$$\begin{aligned} &\textstyle\sum_{j=1}^N \Big(\prod_{j+1}^{i=N} W_i(t)\Big) \dot{W}_j(t) \Big(\prod_1^{i=j-1} W_i(t)\Big) = \\ &\quad -\eta\lambda \textstyle\sum_{j=1}^N \Big(\prod_{j+1}^{i=N} W_i(t)\Big) W_j(t) \Big(\prod_1^{i=j-1} W_i(t)\Big) \\ &\quad -\eta \textstyle\sum_{j=1}^N \Big(\prod_{j+1}^{i=N} W_i(t) \prod_{i=j+1}^N W_i^\top(t)\Big) \cdot \\ &\qquad\qquad \frac{dL^1}{dW}(W_e(t)) \cdot \Big(\textstyle\prod_{i=1}^{j-1} W_i^\top(t) \prod_1^{i=j-1} W_i(t)\Big) \end{aligned}$$

By definition $W_e(t) = \prod_1^{i=N} W_j(t)$, so we can substitute the first two lines above:

$$\begin{aligned} &\dot{W}_e(t) = -\eta\lambda N \cdot W_e(t) \\ &\quad -\eta \sum_{j=1}^N \Big(\textstyle\prod_{j+1}^{i=N} W_i(t) \prod_{i=j+1}^N W_i^\top(t)\Big) \cdot \\ &\qquad\qquad \frac{dL^1}{dW}(W_e(t)) \cdot \Big(\textstyle\prod_{i=1}^{j-1} W_i^\top(t) \prod_1^{i=j-1} W_i(t)\Big) \end{aligned}$$

Finally, plugging in the relations in Equations 23 and 24, the sought-after result is revealed:

$$\begin{aligned} \dot{W}_e(t) &= -\eta\lambda N \cdot W_e(t) \\ &\quad -\eta \sum_{j=1}^N \left[W_e(t)W_e^\top(t)\right]^{\frac{N-j}{N}} \cdot \\ &\qquad\qquad \frac{dL^1}{dW}(W_e(t)) \cdot \left[W_e^\top(t)W_e(t)\right]^{\frac{j-1}{N}} \end{aligned}$$

□

Lemma 1. *Let $I \subset \mathbb{R}$ be a connected interval, and let $f, g : I \to \mathbb{R}$ be differentiable functions. Suppose that there exists a constant $\alpha \geq 0$ for which:*

$$\dot{f}(t) + \alpha \cdot f(t) = \dot{g}(t) + \alpha \cdot g(t) \quad , \forall t \in I$$

Then, if f and g assume the same value at some $t_0 \in I$ (interior or boundary), they must coincide along the entire interval, i.e. *it must hold that $f(t) = g(t)$ for all $t \in I$.*

Proof. Define $h := f - g$. h is a differentiable function from I to $\mathbb{R}$, and we have:

$$\dot{h}(t) = -\alpha \cdot h(t) \quad , \forall t \in I \tag{25}$$

We know that $h(t_0) = 0$ for some $t_0 \in I$, and would like to show that $h(t) = 0 \ \forall t \in I$. Assume by contradiction that this is not the case, so there exists $t_2 \in I$ for which $h(t_2) \neq 0$. Without loss of generality, suppose that $h(t_2) > 0$, and that $t_2 > t_0$. Let S be the zero set of h, *i.e.* $S := \{t \in I : h(t) = 0\}$. Since h is continuous in I, S is topologically closed, therefore its intersection with the interval $[t_0, t_2]$ is compact. Denote by t_1 the maximal element in this intersection, and consider the interval $J := [t_1, t_2] \subset I$. By construction, h is positive along J, besides on the endpoint t_1 where it assumes the value of zero. For $t_1 < t \leq t_2$, we may solve as follows the differential equation of h (Equation 25):

$$\frac{\dot{h}(t)}{h(t)} = -\alpha \quad \Longrightarrow \quad h(t) = \beta e^{-\alpha t}$$

where β is the positive constant defined by $h(t_2) = \beta e^{-\alpha t_2}$. Since in particular h is bounded away from zero on $(t_1, t_2]$, and assumes zero at t_1, we obtain a contradiction to its continuity. This completes the proof. □

A.2. Proof of Claim 1

Our proof relies on the *Kronecker product* operation for matrices. For arbitrary matrices A and B of sizes $m_a \times n_a$ and $m_b \times n_b$ respectively, the Kronecker product $A \odot B$ is defined to be the following block matrix:

$$A \odot B := \begin{bmatrix} a_{11} \cdot B & \cdots & a_{1n_a} \cdot B \\ \vdots & \ddots & \vdots \\ a_{m_a 1} \cdot B & \cdots & a_{m_a n_a} \cdot B \end{bmatrix} \in \mathbb{R}^{m_a m_b, n_a n_b} \tag{26}$$

where a_{ij} stands for the element in row i and column j of A. The Kronecker product admits numerous useful properties. We will employ the following:

- If A and B are matrices such that the matrix product AB is defined, then:

$$\begin{aligned} vec(AB) &= (B^\top \odot I_{r_A}) \cdot vec(A) \\ &= (I_{c_B} \odot A) \cdot vec(B) \end{aligned} \tag{27}$$

  where I_{r_A} and I_{c_B} are the identity matrices whose sizes correspond, respectively, to the number of rows in A and the number of columns in B. $vec(\cdot)$ here, as in claim statement, stands for matrix vectorization in column-first order.

- If A_1, A_2, B_1 and B_2 are matrices such that the matrix products $A_1 B_1$ and $A_2 B_2$ are defined, then:

$$(A_1 \odot A_2)(B_1 \odot B_2) = (A_1 B_1) \odot (A_2 B_2) \tag{28}$$

- For any matrices A and B:

$$(A \odot B)^\top = A^\top \odot B^\top \tag{29}$$

- Equation 28 and 29 imply, that if A and B are some orthogonal matrices, so is $A \odot B$:

$$\begin{aligned} &A^\top = A^{-1} \ \wedge \ B^\top = B^{-1} \\ &\qquad \Longrightarrow \ (A \odot B)^\top = (A \odot B)^{-1} \end{aligned} \tag{30}$$

With the Kronecker product in place, we proceed to the actual proof. It suffices to show that vectorizing:

$$\sum_{j=1}^{N} \left[W_e^{(t)} (W_e^{(t)})^\top \right]^{\frac{j-1}{N}} \cdot \frac{dL^1}{dW}(W_e^{(t)}) \cdot \left[(W_e^{(t)})^\top W_e^{(t)} \right]^{\frac{N-j}{N}}$$

yields:

$$P_{W_e^{(t)}} \cdot vec\left(\frac{dL^1}{dW}(W_e^{(t)}) \right)$$

where $P_{W_e^{(t)}}$ is the preconditioning matrix defined in claim statement. For notational conciseness, we hereinafter omit the iteration index t, and simply write W_e instead of $W_e^{(t)}$.

Let I_d and I_k be the identity matrices of sizes $d \times d$ and $k \times k$ respectively. Utilizing the properties of the Kronecker product, we have:

$$\begin{aligned} &vec\left(\sum_{j=1}^{N} \left[W_e W_e^\top \right]^{\frac{j-1}{N}} \frac{dL^1}{dW}(W_e) \left[W_e^\top W_e \right]^{\frac{N-j}{N}} \right) \\ &= \sum_{j=1}^{N} \left(I_d \odot \left[W_e W_e^\top \right]^{\frac{j-1}{N}} \right) \cdot \\ &\qquad \left(\left[W_e^\top W_e \right]^{\frac{N-j}{N}} \odot I_k \right) \cdot vec\left(\frac{dL^1}{dW}(W_e) \right) \\ &= \sum_{j=1}^{N} \left(\left[W_e^\top W_e \right]^{\frac{N-j}{N}} \odot \left[W_e W_e^\top \right]^{\frac{j-1}{N}} \right) vec\left(\frac{dL^1}{dW}(W_e) \right) \end{aligned}$$

The first equality here makes use of Equation 27, and the second of Equation 28. We will show that the matrix:

$$Q := \sum_{j=1}^{N} \left[W_e^\top W_e \right]^{\frac{N-j}{N}} \odot \left[W_e W_e^\top \right]^{\frac{j-1}{N}} \tag{31}$$

meets the characterization of P_{W_e}, thereby completing the proof. Let:

$$W_e = UDV^\top$$

be a singular value decomposition, *i.e.* $U \in \mathbb{R}^{k,k}$ and $V \in \mathbb{R}^{d,d}$ are orthogonal matrices, and D is a rectangular-diagonal matrix holding (non-negative) singular values on its diagonal. Plug this into the definition of Q (Equation 31):

$$\begin{aligned}
Q &= \sum_{j=1}^{N} \left[VD^\top DV^\top\right]^{\frac{N-j}{N}} \odot \left[UDD^\top U^\top\right]^{\frac{j-1}{N}} \\
&= \sum_{j=1}^{N} \left(V\left[D^\top D\right]^{\frac{N-j}{N}} V^\top\right) \odot \left(U\left[DD^\top\right]^{\frac{j-1}{N}} U^\top\right) \\
&= \sum_{j=1}^{N} (V \odot U)\left(\left[D^\top D\right]^{\frac{N-j}{N}} \odot \left[DD^\top\right]^{\frac{j-1}{N}}\right)(V^\top \odot U^\top) \\
&= (V \odot U)\left(\sum_{j=1}^{N} \left[D^\top D\right]^{\frac{N-j}{N}} \odot \left[DD^\top\right]^{\frac{j-1}{N}}\right)(V \odot U)^\top
\end{aligned}$$

The third equality here is based on the relation in Equation 28, and the last equality is based on Equation 29. Denoting:

$$O := V \odot U \tag{32}$$

$$\Lambda := \sum_{j=1}^{N} \left[D^\top D\right]^{\frac{N-j}{N}} \odot \left[DD^\top\right]^{\frac{j-1}{N}} \tag{33}$$

we have:

$$Q = O\Lambda O^\top \tag{34}$$

Now, since by definition U and V are orthogonal, O is orthogonal as well (follows from the relation in Equation 30). Additionally, the fact that D is rectangular-diagonal implies that the square matrix Λ is also diagonal. Equation 34 is thus an orthogonal eigenvalue decomposition of Q. Finally, denote the columns of U (left singular vectors of W_e) by $\mathbf{u}_1 \ldots \mathbf{u}_k$, those of V (right singular vectors of W_e) by $\mathbf{v}_1 \ldots \mathbf{v}_d$, and the diagonal elements of D (singular values of W_e) by $\sigma_1 \ldots \sigma_{\max\{k,d\}}$ (by definition $\sigma_r = 0$ if $r > \min\{k, d\}$). The definitions in Equations 32 and 33 imply that the columns of O are:

$$vec(\mathbf{u}_r \mathbf{v}_{r'}^\top) \quad , r = 1 \ldots k \ , \ r' = 1 \ldots d$$

with corresponding diagonal elements in Λ being:

$$\sum\nolimits_{j=1}^{N} \sigma_r^{2\frac{N-j}{N}} \sigma_{r'}^{2\frac{j-1}{N}} \quad , r = 1 \ldots k \ , \ r' = 1 \ldots d$$

We conclude that Q indeed meets the characterization of P_{W_e} in claim statement. This completes the proof.

□

A.3. Proof of Claim 2

We disregard the trivial case $N = 1$, as well as the scenario $W_e^{(t)} = 0$ (both lead Equations 10 and 12 to equate). Omitting the iteration index t from our notation, it suffices to show that:

$$\begin{aligned}
&\sum_{j=1}^{N} \left[W_e W_e^\top\right]^{\frac{j-1}{N}} \cdot \frac{dL^1}{dW}(W_e) \cdot \left[W_e^\top W_e\right]^{\frac{N-j}{N}} = \\
&\|W_e\|_2^{2-\frac{2}{N}} \left(\tfrac{dL^1}{dW}(W_e) + (N-1)Pr_{W_e}\left\{\tfrac{dL^1}{dW}(W_e)\right\}\right)
\end{aligned} \tag{35}$$

where $Pr_{W_e}\{\cdot\}$ is the projection operator defined in claim statement (Equation 13), and we recall that by assumption $k = 1$ ($W_e \in \mathbb{R}^{1,d}$). $\left[W_e W_e^\top\right]^{\frac{j-1}{N}}$ is a scalar, equal to $\|W_e\|_2^{2\frac{j-1}{N}}$ for every $j = 1 \ldots N$. $\left[W_e^\top W_e\right]^{\frac{N-j}{N}}$ on the other hand is a $d \times d$ matrix, by definition equal to identity for $j = N$, and otherwise, for $j = 1 \ldots N-1$, it is equal to $\|W_e\|_2^{2\frac{N-j}{N}} \left({}^{W_e}/_{\|W_e\|_2}\right)^\top \left({}^{W_e}/_{\|W_e\|_2}\right)$. Plugging these equalities into the first line of Equation 35 gives:

$$\begin{gathered}
\sum_{j=1}^{N} \left[W_e W_e^\top\right]^{\frac{j-1}{N}} \frac{dL^1}{dW}(W_e) \left[W_e^\top W_e\right]^{\frac{N-j}{N}} = \\
\sum_{j=1}^{N-1} \|W_e\|_2^{2\frac{j-1}{N}} \frac{dL^1}{dW}(W_e) \|W_e\|_2^{2\frac{N-j}{N}} \left(\tfrac{W_e}{\|W_e\|_2}\right)^\top \left(\tfrac{W_e}{\|W_e\|_2}\right) \\
+ \|W_e\|_2^{2\frac{N-1}{N}} \cdot \frac{dL^1}{dW}(W_e) = \\
(N-1)\|W_e\|_2^{2\frac{N-1}{N}} \frac{dL^1}{dW}(W_e) \left(\tfrac{W_e}{\|W_e\|_2}\right)^\top \left(\tfrac{W_e}{\|W_e\|_2}\right) \\
+ \|W_e\|_2^{2\frac{N-1}{N}} \cdot \frac{dL^1}{dW}(W_e)
\end{gathered}$$

The latter expression is precisely the second line of Equation 35, thus our proof is complete.

□

A.4. Proof of Theorem 2

Our proof relies on elementary differential geometry: curves, arc length and line integrals (see Chapters 8 and 9 in Buck (2003)).

Let $\mathcal{U} \subset \mathbb{R}^{1,d}$ be a neighborhood of $W = 0$ (*i.e.* an open set that includes this point) on which $\frac{dL^1}{dW}$ is continuous ($\mathcal{U}$ exists by assumption). It is not difficult to see that $F(\cdot)$ (Equation 14) is continuous on $\mathcal{U}$ as well. The strategy of our proof will be to show that $F(\cdot)$ does not admit the *gradient theorem* (also known as the *fundamental theorem for line integrals*). According to the theorem, if $h : \mathcal{U} \to \mathbb{R}$ is a continuously differentiable function, and Γ is a piecewise smooth curve lying in $\mathcal{U}$ with start-point γ_s and end-point γ_e, then:

$$\int_\Gamma \frac{dh}{dW} = h(\gamma_e) - h(\gamma_s)$$

In words, the line integral of the gradient of h over Γ, is equal to the difference between the value taken by h at the end-point of Γ, and that taken at the start-point. A direct implication of the theorem is that if Γ is closed ($\gamma_e = \gamma_s$), the line integral vanishes:

$$\oint_\Gamma \frac{dh}{dW} = 0$$

We conclude that if $F(\cdot)$ is the gradient field of some function, its line integral over any closed (piecewise smooth) curve lying in $\mathcal{U}$ must vanish. We will show that this is not the case.

For notational conciseness we hereinafter identify $\mathbb{R}^{1,d}$ and $\mathbb{R}^d$, so in particular $\mathcal{U}$ is now a subset of $\mathbb{R}^d$. To further simplify, we omit the subindex from the Euclidean norm, writing $\|\cdot\|$ instead of $\|\cdot\|_2$. Given an arbitrary continuous vector field $\phi : \mathcal{U} \to \mathbb{R}^d$, we define a respective (continuous) vector field as follows:

$$F_\phi : \mathcal{U} \to \mathbb{R}^d$$

$$F_\phi(\mathbf{w}) = \tag{36}$$

$$\begin{cases} \|\mathbf{w}\|^{2-\frac{2}{N}}\Big(\phi(\mathbf{w}) + (N-1)\Big\langle \phi(\mathbf{w}), \frac{\mathbf{w}}{\|\mathbf{w}\|}\Big\rangle \frac{\mathbf{w}}{\|\mathbf{w}\|}\Big) & , \mathbf{w} \neq \mathbf{0} \\ \mathbf{0} & , \mathbf{w} = \mathbf{0} \end{cases}$$

Notice that for $\phi = \frac{dL^1}{dW}$, we get exactly the vector field $F(\cdot)$ defined in theorem statement (Equation 14) – the subject of our inquiry. As an operator on (continuous) vector fields, the mapping $\phi \mapsto F_\phi$ is linear.[5] This, along with the linearity of line integrals, imply that for any piecewise smooth curve Γ lying in $\mathcal{U}$, the functional $\phi \mapsto \int_\Gamma F_\phi$, a mapping of (continuous) vector fields to scalars, is linear. Lemma 2 below provides an upper bound on this linear functional in terms of the length of Γ, its maximal distance from origin, and the maximal norm ϕ takes on it.

In light of the above, to show that $F(\cdot)$ contradicts the gradient theorem, thereby completing the proof, it suffices to find a closed (piecewise smooth) curve Γ for which the linear functional $\phi \mapsto \oint_\Gamma F_\phi$ does not vanish at $\phi = \frac{dL^1}{dW}$. By assumption $\frac{dL^1}{dW}(W{=}0) \neq 0$, and so we may define the unit vector in the direction of $\frac{dL^1}{dW}(W{=}0)$:

$$\mathbf{e} := \frac{\frac{dL^1}{dW}(W{=}0)}{\|\frac{dL^1}{dW}(W{=}0)\|} \in \mathbb{R}^d \tag{37}$$

Let R be a positive constant small enough such that the Euclidean ball of radius R around the origin is contained in $\mathcal{U}$. Let r be a positive constant smaller than R. Define $\Gamma_{r,R}$ to be a curve as follows (see illustration in Figure 1):[6]

$$\Gamma_{r,R} := \Gamma^1_{r,R} \;\rightarrow\; \Gamma^2_{r,R} \;\rightarrow\; \Gamma^3_{r,R} \;\rightarrow\; \Gamma^4_{r,R} \tag{38}$$

where:

- $\Gamma^1_{r,R}$ is the line segment from $-R \cdot \mathbf{e}$ to $-r \cdot \mathbf{e}$.
- $\Gamma^2_{r,R}$ is a geodesic on the sphere of radius r, starting from $-r \cdot \mathbf{e}$ and ending at $r \cdot \mathbf{e}$.
- $\Gamma^3_{r,R}$ is the line segment from $r \cdot \mathbf{e}$ to $R \cdot \mathbf{e}$.
- $\Gamma^4_{r,R}$ is a geodesic on the sphere of radius R, starting from $R \cdot \mathbf{e}$ and ending at $-R \cdot \mathbf{e}$.

$\Gamma_{r,R}$ is a piecewise smooth, closed curve that fully lies within $\mathcal{U}$. Consider the linear functional it induces: $\phi \mapsto \oint_{\Gamma_{r,R}} F_\phi$. We will evaluate this functional on $\phi = \frac{dL^1}{dW}$. To do so, we decompose the latter as follows:

$$\tfrac{dL^1}{dW}(\cdot) = c \cdot \mathbf{e}(\cdot) + \xi(\cdot) \tag{39}$$

where:

- c is a scalar equal to $\|\frac{dL^1}{dW}(W{=}0)\|$.
- $\mathbf{e}(\cdot)$ is a vector field returning the constant $\mathbf{e}$ (Equation 37).
- $\xi(\cdot)$ is a vector field returning the values of $\frac{dL^1}{dW}(\cdot)$ shifted by the constant $-\frac{dL^1}{dW}(W{=}0)$. It is continuous on $\mathcal{U}$ and vanishes at the origin.

Applying Lemma 2 to ξ over $\Gamma_{r,R}$ gives:

$$\begin{aligned} \left|\oint_{\Gamma_{r,R}} F_\xi\right| &\leq N \cdot len(\Gamma_{r,R}) \cdot \max_{\gamma \in \Gamma_{r,R}} \|\gamma\|^{2-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma_{r,R}} \|\xi(\gamma)\| \\ &= N \cdot (\pi r + \pi R + 2(R-r)) \cdot R^{2-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma_{r,R}} \|\xi(\gamma)\| \\ &\leq N \cdot 2\pi \cdot R^{3-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma_{r,R}} \|\xi(\gamma)\| \end{aligned}$$

On the other hand, by Lemma 3:

$$\oint_{\Gamma_{r,R}} F_{\mathbf{e}} = \left(\frac{2N}{3 - {}^2/_N} - 2\right)\left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}}\right)$$

[5] For any $\phi_1, \phi_2 : \mathcal{U} \to \mathbb{R}^d$ and $c \in \mathbb{R}$, it holds that $F_{\phi_1+\phi_2} = F_{\phi_1} + F_{\phi_2}$ and $F_{c\cdot\phi_1} = c \cdot F_{\phi_1}$.

[6] The proof would have been slightly simplified had we used a curve that passes directly through the origin. We avoid this in order to emphasize that the result is not driven by some point-wise singularity (the origin received special treatment in the definition of $F(\cdot)$ – see Equations 14 and 13).

The linearity of the functional $\phi \mapsto \oint_{\Gamma_{r,R}} F_\phi$, along with Equation 39, then imply:

$$\begin{aligned}\oint_{\Gamma_{r,R}} F_{\frac{dL^1}{dW}} &= c \cdot \oint_{\Gamma_{r,R}} F_{\mathbf{e}} + \oint_{\Gamma_{r,R}} F_\xi \\ &\geq c \cdot \left(\frac{2N}{3 - 2/N} - 2 \right) \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right) \\ &\quad - N \cdot 2\pi \cdot R^{3-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma_{r,R}} \|\xi(\gamma)\|\end{aligned}$$

We will show that for proper choices of R and r, the lower bound above is positive. $\Gamma_{r,R}$ will then be a piecewise smooth closed curve lying in $\mathcal{U}$, for which the functional $\phi \mapsto \oint_{\Gamma_{r,R}} F_\phi$ does not vanish at $\phi = \frac{dL^1}{dW}$. As stated, this will imply that $F(\cdot)$ violates the gradient theorem, thereby concluding our proof.

All that is left is to affirm that the expression:

$$\begin{gathered} c \cdot \left(\tfrac{2N}{3-2/N} - 2 \right) \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right) \\ -N \cdot 2\pi \cdot R^{3-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma_{r,R}} \|\xi(\gamma)\| \end{gathered}$$

can indeed be made positive with proper choices of R and r. Recall that:

- $N > 2$ by assumption; implies $\frac{2N}{3-2/N} - 2 > 0$.
- R is any positive constant small enough such that the ball of radius R around the origin is contained in $\mathcal{U}$.
- r is any positive constant smaller than R.
- $\Gamma_{r,R}$ is a curve whose points are all within distance R from the origin.
- $c = \|\frac{dL^1}{dW}(W{=}0)\|$ – positive by assumption.
- $\xi(\cdot)$ is a vector field that is continuous on $\mathcal{U}$ and vanishes at the origin.

The following procedure gives R and r as required:

- Set r to follow R such that: $r^{3-\frac{2}{N}} = 0.5 \cdot R^{3-\frac{2}{N}}$.
- Choose $\epsilon > 0$ for which $0.5c \left(\frac{2N}{3-\frac{2}{N}} - 2 \right) - 2\pi N \epsilon > 0$.
- Set R to be small enough such that $\|\xi(\mathbf{w})\| \leq \epsilon$ for any point $\mathbf{w}$ within distance R from the origin.

The proof is complete.

□

Lemma 2. *Let $\phi : \mathcal{U} \to \mathbb{R}^d$ be a continuous vector field, and let Γ be a piecewise smooth curve lying in $\mathcal{U}$. Consider the (continuous) vector field $F_\phi : \mathcal{U} \to \mathbb{R}^d$ defined in Equation 36. The line integral of the latter over Γ is bounded as follows:*

$$\left| \int_\Gamma F_\phi \right| \leq N \cdot len(\Gamma) \cdot \max_{\gamma \in \Gamma} \|\gamma\|^{2-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma} \|\phi(\gamma)\|$$

where $len(\Gamma)$ is the arc length of Γ, and $\gamma \in \Gamma$ refers to a point lying on the curve.

Proof. We begin by noting that the use of $\max$ (as opposed to $\sup$) in stated upper bound is appropriate, since under our definition of a curve (adopted from Buck (2003)), points lying on it constitute a compact set. This subtlety is of little importance – one may as well replace $\max$ by $\sup$, and the lemma would still serve its purpose.

It is not difficult to see that for any $\mathbf{w} \in \mathcal{U}$, $\mathbf{w} \neq 0$:

$$\begin{aligned}&\|F_\phi(\mathbf{w})\| = \|\mathbf{w}\|^{2-\frac{2}{N}} \left\| \phi(\mathbf{w}) + (N-1) \left\langle \phi(\mathbf{w}), \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\rangle \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\| \\ &\leq \|\mathbf{w}\|^{2-\frac{2}{N}} \left(\|\phi(\mathbf{w})\| + (N-1) \left| \left\langle \phi(\mathbf{w}), \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\rangle \right| \cdot \left\| \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\| \right) \\ &= \|\mathbf{w}\|^{2-\frac{2}{N}} \left(\|\phi(\mathbf{w})\| + (N-1) \left| \left\langle \phi(\mathbf{w}), \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\rangle \right| \right) \\ &\leq \|\mathbf{w}\|^{2-\frac{2}{N}} (\|\phi(\mathbf{w})\| + (N-1)\|\phi(\mathbf{w})\|) \\ &\leq N \|\mathbf{w}\|^{2-\frac{2}{N}} \|\phi(\mathbf{w})\|\end{aligned}$$

Trivially, $\|F_\phi(\mathbf{w})\| \leq N \|\mathbf{w}\|^{2-\frac{2}{N}} \|\phi(\mathbf{w})\|$ holds for $\mathbf{w}{=}0$ as well. The sought-after result now follows from the properties of line integrals:

$$\begin{aligned}\left| \int_\Gamma F_\phi \right| &\leq \int_\Gamma \|F_\phi\| \leq \int_\Gamma N \|\mathbf{w}\|^{2-\frac{2}{N}} \|\phi(\mathbf{w})\| \\ &\leq N \cdot len(\Gamma) \cdot \max_{\gamma \in \Gamma} \|\gamma\|^{2-\frac{2}{N}} \cdot \max_{\gamma \in \Gamma} \|\phi(\gamma)\|\end{aligned}$$

□

Lemma 3. *Let $\mathbf{e}$ be a unit vector, let $\Gamma_{r,R}$ be a piecewise smooth closed curve as specified in Equation 38 and the text thereafter, and let $\phi \mapsto F_\phi$ be the operator on continuous vector fields defined by Equation 36. Overloading notation by regarding $\mathbf{e}(\cdot) \equiv \mathbf{e}$ as a constant vector field, it holds that:*

$$\oint_{\Gamma_{r,R}} F_{\mathbf{e}} = \left(\frac{2N}{3 - 2/N} - 2 \right) \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right)$$

Proof. We compute the line integral by decomposing $\Gamma_{r,R}$ into its smooth components $\Gamma^1_{r,R} \ldots \Gamma^4_{r,R}$:

$$\oint_{\Gamma_{r,R}} F_{\mathbf{e}} = \int_{\Gamma^1_{r,R}} F_{\mathbf{e}} + \int_{\Gamma^2_{r,R}} F_{\mathbf{e}} + \int_{\Gamma^3_{r,R}} F_{\mathbf{e}} + \int_{\Gamma^4_{r,R}} F_{\mathbf{e}} \qquad (40)$$

Starting from $\Gamma^1_{r,R}$, notice that for every point $\mathbf{w}$ lying on this curve: $\langle \mathbf{e}, \frac{\mathbf{w}}{\|\mathbf{w}\|} \rangle \frac{\mathbf{w}}{\|\mathbf{w}\|} = \mathbf{e}$. Therefore:

$$\int_{\Gamma^1_{r,R}} F_{\mathbf{e}} = \int_{\Gamma^1_{r,R}} \|\mathbf{w}\|^{2-\frac{2}{N}} (\mathbf{e}+(N-1)\mathbf{e}) = N \int_{\Gamma^1_{r,R}} \|\mathbf{w}\|^{2-\frac{2}{N}} \mathbf{e}$$

The line integral on the right translates into a simple univariate integral:

$$\int_{\Gamma^1_{r,R}} \|\mathbf{w}\|^{2-\frac{2}{N}} \mathbf{e} = \int_{-R}^{-r} |\rho|^{2-\frac{2}{N}} d\rho = \int_{r}^{R} \rho^{2-\frac{2}{N}} d\rho$$
$$= \frac{1}{3-{}^2\!/_N} \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right)$$

We thus have:

$$\int_{\Gamma^1_{r,R}} F_{\mathbf{e}} = \frac{N}{3-{}^2\!/_N} \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right) \tag{41}$$

Turning to $\Gamma^2_{r,R}$, note that for any point $\mathbf{w}$ along this curve $\|\mathbf{w}\|^{2-\frac{2}{N}} = r^{2-\frac{2}{N}}$, and $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is perpendicular to the direction of motion. This implies:

$$\int_{\Gamma^2_{r,R}} F_{\mathbf{e}} = r^{2-\frac{2}{N}} \int_{\Gamma^2_{r,R}} \mathbf{e}$$

The line integral $\int_{\Gamma^2_{r,R}} \mathbf{e}$ is simply equal to the progress $\Gamma^2_{r,R}$ makes in the direction of $\mathbf{e}$, which is $2r$. Accordingly:

$$\int_{\Gamma^2_{r,R}} F_{\mathbf{e}} = r^{2-\frac{2}{N}} \cdot 2r = 2r^{3-\frac{2}{N}} \tag{42}$$

As for $\Gamma^3_{r,R}$ and $\Gamma^4_{r,R}$, their line integrals may be computed similarly to those of $\Gamma^1_{r,R}$ and $\Gamma^2_{r,R}$ respectively. Such computations yield:

$$\int_{\Gamma^3_{r,R}} F_{\mathbf{e}} = \frac{N}{3-{}^2\!/_N} \left(R^{3-\frac{2}{N}} - r^{3-\frac{2}{N}} \right) \tag{43}$$

$$\int_{\Gamma^4_{r,R}} F_{\mathbf{e}} = -2R^{3-\frac{2}{N}} \tag{44}$$

Combining Equation 40 with Equations 41, 42, 43 and 44, we obtain the desired result. □

B. A Concrete Acceleration Bound

In Section 7 we illustrated qualitatively, on a family of very simple hypothetical learning problems, the potential of overparameterization (use of depth-N linear network in place of classic linear model) to accelerate optimization. In this appendix we demonstrate how the illustration can be made formal, by considering a special case and deriving a concrete bound on the acceleration.

In the context of Section 7, we will treat the setting of $p = 4$ (ℓ_4 loss) and $N = 2$ (depth-2 network). We will also assume, in accordance with the problem being ill-conditioned – $y_1 \gg y_2$, that initialization values are ill-conditioned as well, and in particular $\epsilon_1/\epsilon_2 \approx y_1/y_2$, where $\epsilon_i := |w_i^{(0)}|$. An additional assumption we make is that y_2 is on the order of 1, and thus the near-zero initialization of w_1 and w_2 implies $y_2 \gg \epsilon_1, \epsilon_2$. Finally, we assume that $\epsilon_1 y_1 \gg 1$.

As shown in Section 7, under gradient descent, w_1 and w_2 move independently, and to prevent divergence, the learning rate must satisfy $\eta < \min\{2/y_1^{p-2}, 2/y_2^{p-2}\}$. In our setting, this translates to (GD below stands for gradient descent):

$$\eta^{GD} < 2/y_1^2 \tag{45}$$

For w_2, the optimal learning rate (convergence in a single step) is $1/y_2^2$, and the constraint above will lead to very slow convergence (see Equation 15 and its surrounding text).

Suppose now that we optimize via overparameterization, *i.e.* with the update rule in Equation 12 (single output). In our particular setting (recall, in addition to the above, that we omitted weight decay for simplicity – $\lambda = 0$), this update rule translates to:

$$\begin{aligned}
&[w_1^{(t+1)}, w_2^{(t+1)}]^\top \mapsfrom [w_1^{(t)}, w_2^{(t)}]^\top \\
&\quad -\eta\Big((w_1^{(t)})^2 + (w_2^{(t)})^2\Big)^{1/2} \cdot [(w_1^{(t)} - y_1)^3, (w_2^{(t)} - y_2)^3]^\top \\
&\quad -\eta\Big((w_1^{(t)})^2 + (w_2^{(t)})^2\Big)^{-1/2} \\
&\qquad \cdot \big(w_1^{(t)}(w_1^{(t)} - y_1)^3 + w_2^{(t)}(w_2^{(t)} - y_2)^3\big) \cdot [w_1^{(t)}, w_2^{(t)}]^\top
\end{aligned} \tag{46}$$

For the first iteration ($t = 0$), replacing $\epsilon_i := |w_i^{(0)}|$, while recalling that $y_1 \gg y_2 \gg \epsilon_1 \gg \epsilon_2$, we obtain:

$$\begin{aligned}
[w_1^{(1)}, w_2^{(1)}]^\top &\approx \eta \cdot \epsilon_1 \cdot [y_1^3, y_2^3]^\top + \eta \cdot y_1^3 \cdot [\epsilon_1, \epsilon_2]^\top \\
&= \eta \cdot [2\epsilon_1 y_1^3, \epsilon_1 y_2^3 + \epsilon_2 y_1^3]^\top
\end{aligned}$$

Set $\eta = 1/2\epsilon_1 y_1^2$. Then $w_1^{(1)} \approx y_1$ and $w_2^{(1)} \approx y_2^3/2y_1^2 + \epsilon_2 y_1/2\epsilon_1$. Our assumptions thus far ($y_1 \gg y_2$ and $\epsilon_1 \gg \epsilon_2$) imply $w_1^{(1)} \gg w_2^{(1)}$. Moreover, since $\epsilon_2/\epsilon_1 \approx y_2/y_1$, it holds that $w_2^{(1)} \in \mathcal{O}(y_2) = \mathcal{O}(1)$. Taking all of this into account, the second iteration ($t = 1$) of the overparameterized update rule (Equation 46) becomes:

$$\begin{aligned}
[w_1^{(2)}, w_2^{(2)}]^\top &\approx [y_1, w_2^{(1)}]^\top \\
&\quad - \frac{1}{2\epsilon_1 y_1}[(w_1^{(1)} - y_1)^3, (w_2^{(1)} - y_2)^3]^\top \\
&\quad - \frac{y_1(w_1^{(1)} - y_1)^3 + w_2^{(1)}(w_2^{(1)} - y_2)^3}{2\epsilon_1 y_1^3}[y_1, w_2^{(1)}]^\top \\
&\approx [y_1, w_2^{(1)} - 1/2\epsilon_1 y_1 \cdot (w_2^{(1)} - y_2)^3]^\top
\end{aligned}$$

In words, w_1 will stay approximately equal to y_1, whereas w_2 will take a step that corresponds to gradient descent with learning rate (OP below stands for overparameterization):

$$\eta^{OP} := 1/2\epsilon_1 y_1 \tag{47}$$

By assumption $\epsilon_1 y_1 \gg 1$ and $y_2 \in \mathcal{O}(1)$, thus $\eta^{OP} < 2/y_2^2$, meaning that w_2 will remain on the order of y_2 (or less). An inductive argument can therefore be applied, and our observation regarding the second iteration ($t = 1$) continues to hold throughout – w_1 is (approximately) fixed at y_1, and w_2 follows steps that correspond to gradient descent with learning rate η^{OP}.

To summarize our findings, we have shown that while standard gradient descent limits w_2 with a learning rate η^{GD} that is at most $2/y_1^2$ (Equation 45), overparameterization can be adjusted to induce on w_2 an implicit gradient descent scheme with learning rate $\eta^{OP} = 1/2\epsilon_1 y_1$ (Equation 47), all while admitting immediate (single-step) convergence for w_1. Since both η^{GD} and η^{OP} are well below $1/y_2^2$, we obtain acceleration by at least $\eta^{OP}/\eta^{GD} > y_1/4\epsilon_1$ (we remind the reader that $y_1 \gg 1$ is the target value of w_1, and $\epsilon_1 \ll 1$ is the magnitude of its initialization).

C. Implementation Details

Below we provide implementation details omitted from our experimental report (Section 8).

C.1. Linear Neural Networks

The details hereafter apply to all of our experiments besides that on the convolutional network (Figure 5-right).

In accordance with our theoretical setup (Section 4), evaluated linear networks did not include bias terms, only weight matrices. The latter were initialized to small values, drawn i.i.d. from a Gaussian distribution with mean zero and standard deviation 0.01. The only exception to this was the setting of identity initialization (Figure 5-left), in which an offset of 1 was added to the diagonal elements of each weight matrix (including those that are not square).

When applying a grid search over learning rates, the values $\{10^{-5}, 5 \cdot 10^{-5}, \ldots, 10^{-1}, 5 \cdot 10^{-1}\}$ were tried. We note that in the case of depth-8 network with standard near-zero initialization (Figure 5-left), all learning rates led either to divergence, or to a failure to converge (vanishing gradients).

For computing optimal ℓ_2 loss (used as an offset in respective convergence plots), we simply solved, in closed form, the corresponding least squares problem. For the optimal ℓ_4 loss, we used `scipy.optimize.minimize` – a numerical optimizer built into SciPy (Jones et al., 2001–), with the default method of BFGS (Nocedal, 1980).

C.2. Convolutional Network

For the experiment on TensorFlow's MNIST convolutional network tutorial, we simply downloaded the code,[7] and introduced two minor changes:

- Hidden dense layer: 3136×512 weight matrix replaced by multiplication of 3136×512 and 512×512 matrices.
- Output layer: 512×10 weight matrix replaced by multiplication of 512×10 and 10×10 matrices.

The newly introduced weight matrices were initialized in the same way as their predecessors (random Gaussian distribution with mean zero and standard deviation 0.1). Besides the above, no change was made. An addition of roughly $250K$ parameters to a $1.6M$-parameter model gave the speedup presented in Figure 5-right.

To rule out the possibility of the speedup resulting from suboptimal learning rates, we reran the experiment with grid search over the latter. The learning rate hardcoded into the tutorial follows an exponentially decaying schedule, with base value 10^{-2}. For both the original and overparameterized models, training was run multiple times, with the base value varying in $\{10^{-5}, 5 \cdot 10^{-5}, \ldots, 10^{-1}, 5 \cdot 10^{-1}\}$. We chose, for each model separately, the configuration giving fastest convergence, and then compared the models one against the other. The observed gap in convergence rates was similar to that in Figure 5-right.

An additional point we set out to examine, is the sensitivity of the speedup to initialization of overparameterized layers. For this purpose, we retrained the overparameterized model multiple times, varying in $\{10^{-3}, 5 \cdot 10^{-3}, \ldots, 10^{-1}, 5 \cdot 10^{-1}\}$ the standard deviation of the Gaussian distribution initializing overparameterized layers (as stated above, this standard deviation was originally set to 10^{-1}). Convergence rates across the different runs were almost identical. In particular, they were all orders of magnitude faster than the convergence rate of the baseline, non-overparameterized model.

[7] `https://github.com/tensorflow/models/tree/master/tutorials/image/mnist`