Literature Review of Optimization in Deep Networks: Implicit Acceleration by Overparameterization

Kashyap Ava

May 5 2024

Abstract

This report investigates the distinct impact of increasing the depth in linear neural networks on the optimization process, independent of improvements in model expressiveness. Traditional views hold that deeper networks enhance expressiveness but complicate optimization. This study challenges these views by focusing on linear neural networks where additional depth leads to overparameterization. It demonstrates that depth can sometimes be an implicit preconditioner, accelerating optimization. Furthermore, it establishes that the acceleration effect of overparameterization is not achievable through the gradients of any regularizer, illustrating a unique advantage of depth in these networks.

1 Introduction

Though deeper neural networks are thought to be more expressive, this has only been theoretically shown for specific learning problems [2]. These networks are more capable of capturing complexity, but they still have to overcome significant optimization challenges, such as the vanishing or exploding gradient problem [3]. This study suggests, counter-intuitively, that deepening the network could paradoxically speed up optimization, akin to the dynamics of the combination of momentum [4] and adaptive regularization [5] methods such as AdaGrad. This acceleration is implicitly achieved by increasing depth alone without further architectural changes.
One critical question in the exploration of neural network depth is distinguishing whether observed improvements in training arise from actual optimization acceleration or merely from enhanced representational power. So, the study focuses on linear neural networks [6], where adding layers leads to overparameterization without altering the model's expressiveness. This overparameterization transforms a single matrix parameter into a product of matrices, effectively simplifying the network architecture to behave like a shallow, single-layer network but with a unique preconditioning mechanism in place. Experimental results show that this approach outperforms naive gradient descent and accelerates beyond well-established methods like AdaGrad [5] and AdaDelta [7]. Furthermore, to confirm the generalizability of these findings, the study empirically demonstrates that replacing traditionally hidden layers with depth-2 linear networks continues to yield acceleration benefits in nonlinear models, indicating that the advantages of overparameterization in speeding up the optimization process are not confined to linear models alone.

2 Related Work

Optimization in deep learning, a central theme in theoretical studies, often addresses critical points across various network types, from linear models like those explored by [8] to nonlinear setups under restrictive assumptions, as discussed by [9]. Specifically, the dynamics of optimization in linear networks have been examined in [10], which analyzed gradient descent through differential equations similar to this approach. These studies, such as [11], which focused on L_2 regression in networks beyond a depth of two layers, suggest that increased depth can modestly decelerate optimization. Intriguingly, we observe that acceleration in l_p regression becomes evident only when $p > 2$. This critical distinction in the behavior of network depth under different regression models elucidates the theoretical divergence from the conclusions reached by [11]. Further, acceleration techniques such as those pioneered by [4] and using preconditioners like the BFGS algorithm, detailed by [12], show significant advancements in optimizing gradients.

3 Example of l_p Regression

For a scalar linear regression with the loss function based on l_p is defined as:

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

where $x \in \mathbb{R}^d$ are the feature vectors, $y \in \mathbb{R}$ the continuous labels, S the training set, and $w \in \mathbb{R}^d$ the parameter vector. To explore the impact of overparameterization, we replace the parameter vector w with a product of a vector $w_1 \in \mathbb{R}^d$ and a scalar $w_2 \in \mathbb{R}$, resulting in the modified loss function:

$$L(w_1, w_2) = \mathbb{E}_{(x,y)\sim S}[(x^\top w_1 w_2 - y)^p]. \tag{2}$$

This overparameterization does not affect the expressiveness of the model but transforms the optimization landscape. Specifically, employing gradient descent on this non-convex objective reveals unique dynamics. The gradients of $L(w)$ and $L(w_1, w_2)$ are given by:

$$\nabla w = \mathbb{E}_{(x,y)\sim S}[(x^\top w - y)^{p-1} x] \tag{3}$$

$$\nabla w_1 = \mathbb{E}_{(x,y)\sim S}[(x^\top w_1 w_2 - y)^{p-1} w_2 x] \tag{4}$$

$$\nabla w_2 = \mathbb{E}_{(x,y)\sim S}[(x^\top w_1 w_2 - y)^{p-1} w_1^\top x] \tag{5}$$

respectively. When gradient descent is applied to $L(w_1, w_2)$ with a small learning rate η, the dynamics of $w = w_1 w_2$ evolve as the following assuming a negligible second-order term ($O(\eta^2)$):

$$w^{(t+1)} = w^{(t)} - \eta(w^{(t)})^2 \nabla w^{(t)} - \eta(w^{(t)})^{-1} \nabla w_2^{(t)} w^{(t)}, \tag{6}$$

Assuming near zero initialization for both w_1 and w_2, the updated value $w^{(t+1)}$ is influenced heavily by a weighted combination of past gradients that can be formally expressed as:

$$w^{(t+1)} = w^{(t)} - \rho(t) \nabla w^{(t)} - \sum_{\tau=1}^{t-1} \mu(t, \tau) \nabla w^{(\tau)}, \tag{7}$$

where $\rho(t) = \eta(w^{(t)})^2$ and $\mu(t, \tau)$ represent time-varying learning rates and momentum coefficients, respectively. This formulation highlights how overparameterization accelerates the optimization of w by integrating an adaptive momentum mechanism, thereby enhancing convergence efficiency through the strategic use of gradient history in a non-convex landscape.

4 Linear Neural Networks

In neural networks, for a predictor ϕ, which is a mapping from X to Y, the overall training loss is defined as $L(\phi) := \frac{1}{m} \sum_{i=1}^m l(\phi(x^{(i)}), y^{(i)})$. When ϕ is from a parametric family $\Phi := \{\phi_\theta : X \to Y \mid \theta \in \Theta\}$, the training loss becomes a function of the parameters, denoted by $L_\Phi(\theta) := \frac{1}{m} \sum_{i=1}^m l(\phi_\theta(x^{(i)}), y^{(i)})$. In the context of a depth-N ($N \geq 2$) linear neural network with hidden widths $n_1, n_2, \ldots, n_{N-1} \in \mathbb{N}$:

$$\Phi_N = \Phi_{n_1 \ldots n_{N-1}} := \{x \mapsto W_N W_{N-1} \cdots W_1 x \mid W_j \in \mathbb{R}^{n_j \times n_{j-1}}, j = 1 \ldots N\} \tag{8}$$

The training loss corresponding to a depth-N network, $L_{\Phi_N}(W_1, \ldots, W_N)$, transforms into:

$$L_N(\cdot) := \text{function from } \mathbb{R}^{n_1 \times n_0} \times \cdots \times \mathbb{R}^{n_N \times n_{N-1}} \text{ to } \mathbb{R}_{\geq 0}. \tag{9}$$

Our primary interest lies in examining how gradient descent behaves when minimizing $L_N(\cdot)$, specifically focusing on whether increasing N can lead to optimization acceleration. It is notable that for any $N \geq 2$,

$$L_N(W_1, \ldots, W_N) = L_1(W_N W_{N-1} \cdots W_1), \tag{10}$$

indicating that the only difference between the training loss of a depth-N network and a depth-1 network lies in replacing a single matrix parameter with a product of N matrices. This setup suggests that if increasing N accelerates convergence; this effect would solely stem from depth-induced overparameterization's favorable properties for optimization.

5 Implicit Dynamics of Gradient Descent

When gradient descent is applied to $L_N(\cdot)$, it takes the form:

$$W_j^{(t+1)} = (1 - \eta\lambda) W_j^{(t)} - \eta \frac{\partial L_N}{\partial W_j}(W_1^{(t)}, \ldots, W_N^{(t)}), \tag{11}$$

for $j = 1, \ldots, N$, where $\eta > 0$ is a learning rate and $\lambda \geq 0$ represents an optional weight decay coefficient, with both parameters held constant over iterations. The end-to-end weight matrix can be defined as $W_e :=$

$W_N W_{N-1} \cdots W_1$. Considering that $L_N(W_1, \ldots, W_N) = L_1(W_e)$ (as per Equation 10), the study focuses on the dynamics of W_e under these gradient updates. For $N = 1$, these dynamics trivially align with standard gradient descent on $L_1(\cdot)$.

To further elucidate the dynamics for $N \geq 2$, assume a sufficiently small learning rate such that $\eta^2 \approx 0$, allowing to model the updates with a system of differential equations:

$$\dot{W}_j(t) = -\eta\lambda W_j(t) - \eta \frac{\partial L_N}{\partial W_j}(W_1(t), \ldots, W_N(t)), \tag{12}$$

for $j = 1, \ldots, N$, where t is now a continuous time index, and $\dot{W}_j(t)$ denotes the time derivative of W_j. This framework facilitates the application of continuous-time dynamics, commonly used to model the trajectories of optimization algorithms under the assumption of infinitesimally small step sizes [13].

Assuming that the initial conditions for the weight matrices $W_1, \ldots, W_N$ at time t_0 ensure normality (i.e., $W_{j+1}^\top(t_0) W_{j+1}(t_0) = W_j(t_0) W_j^\top(t_0)$ for $j = 1, \ldots, N-1$), the dynamics of W_e are governed by:

$$\dot{W}_e(t) = -\eta\lambda N \cdot W_e(t) - \eta \sum_{j=1}^{N} \left[W_e(t) W_e^\top(t)\right]^{\frac{j-1}{N}} \frac{\partial L_1}{\partial W_e}(W_e(t)) \left[W_e^\top(t) W_e(t)\right]^{\frac{N-j}{N}}. \tag{13}$$

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

The proof involves computing the derivative of $L_N(\cdot)$ with respect to W_j. By chain rule we get:

$$\frac{\partial L_N}{\partial W_j} = \prod_{i=j+1}^{N} W_i^\top \cdot \frac{\partial L_1}{\partial W_e}(W_e) \cdot \prod_{i=1}^{j-1} W_i^\top \tag{14}$$

where

$$\prod_{j=a}^{b} W_j := W_b W_{b-1} \cdots W_a, \quad \prod_{j=a}^{b} W_j^\top := W_a^\top W_{a+1}^\top \cdots W_b^\top$$

Plugging this into the differential equation of gradient descent (Equation 11), we get:

$$\dot{W}_j(t) = -\eta\lambda W_j(t) - \eta \left(\prod_{i=j+1}^{N} W_i^\top(t) \right) \frac{\partial L_1}{\partial W_e}(W_e(t)) \left(\prod_{i=1}^{j-1} W_i^\top(t) \right) \tag{15}$$

For $j = 1, \ldots, N-1$, multiplying the jth equation by $W_j^\top(t)$ from the right, and the $(j+1)$th equation by $W_{j+1}(t)$ from the left. Then taking the transpose of these equations and summing them for every $j = 1, \ldots, N-1$, and based on the assumption of the initial conditions this translates to:

$$W_{j+1}^\top(t) W_{j+1}(t) = W_j(t) W_j^\top(t). \tag{16}$$

Considering the eigen value decomposition of $W_j(t)$ and the property by definition that Σ_{j+1} and Σ_j have non-increasing diagonals, it must hold that:

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

Assuming $\rho_1 > \rho_2 > \ldots > \rho_m \geq 0$ to be the distinct eigenvalues, with corresponding multiplicities $d_1, d_2, \ldots, d_m \in \mathbb{N}$. We may write:

$$\Sigma_{j+1}^\top \Sigma_{j+1} = \Sigma_j \Sigma_j^\top = \operatorname{diag}(\rho_1 I_{d_1}, \ldots, \rho_m I_{d_m}) \tag{17}$$

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

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

$O_{j,r}$ 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. Utilizing these results, the concatenation of weight matrices thus simplify as follows:

$$\prod_{j=N}^{1} W_i(t) \prod_{i=N}^{j} W_i^\top(t) = U_N \cdot \operatorname{diag}((\rho_1)^{N-j+1} I_{d_1}, \ldots, (\rho_m)^{N-j+1} I_{d_m}) \cdot U_N^\top \tag{18}$$

$$\prod_{i=1}^{j} W_i^\top(t) \prod_{i=j}^{1} W_i(t) = V_1 \cdot \mathrm{diag}((\rho_1)^j I_{d_1}, \ldots, (\rho_m)^j I_{d_m}) \cdot V_1^\top \tag{19}$$

where the orthogonality of $O_{j,r}$, and the obvious fact that it commutes with I_{d_r} was used. Rewriting these equations based on the definition $W_e(t) = \prod_{j=N}^{1} W_j(t)$:

$$W_e(t)W_e^\top(t) = U_N \cdot \mathrm{diag}((\rho_1)^N I_{d_1}, \ldots, (\rho_m)^N I_{d_m}) \cdot U_N^\top \tag{20}$$

$$W_e^\top(t)W_e(t) = V_1 \cdot \mathrm{diag}((\rho_1)^N I_{d_1}, \ldots, (\rho_m)^N I_{d_m}) \cdot V_1^\top \tag{21}$$

Therefore, it follows that for every $j = 1 \ldots N$:

$$\prod_{i=j}^{N} W_i(t) \prod_{i=j}^{N} W_i^\top(t) = \left[W_e(t)W_e^\top(t)\right]^{\frac{N-j+1}{N}} \tag{22}$$

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

where $[\cdot]^{\frac{N-j+1}{N}}$ and $[\cdot]^{\frac{j}{N}}$ stand for fractional power operators defined over positive semi-definite matrices. By plugging these equations 22 and 23 in equation 16, we get the eqaution 13. Translating the continuous dynamics of Equation 13 back to discrete time, we obtain the sought-after update rule for the end-to-end weight matrix:

$$W_e^{(t+1)} = (1 - \eta\lambda N)W_e^{(t)} - \eta \sum_{j=1}^{N} \left[W_e^{(t)} \left(W_e^{(t)}\right)^\top\right]^{\frac{j-1}{N}} \frac{\partial L_1}{\partial W_e}\left(W_e^{(t)}\right) \left[\left(W_e^{(t)}\right)^\top W_e^{(t)}\right]^{\frac{N-j}{N}} \tag{24}$$

The above update rule has two primary assumptions: first that learning rate η is small and second, that weights are initialized on par with $W_{j+1}^\top(t_0)W_{j+1}(t_0) = W_j(t_0)W_j^\top(t_0)$, which will approximately be the case even if initialization values are close enough to zero. Equation 24 is similar to a gradient descent over $L_1(\cdot)$ — training loss corresponding to a depth-1 network. The only difference lies in the fact that the gradient is subject to transformation before being used. This motivated the researchers to look at the vector arrangement of equation 24 for further interpretation. This was achieved by using the properties of Kronecker product which gave the following result:

$$\mathrm{vec}\left(\sum_{j=1}^{N} \left[W_e W_e^\top\right]^{\frac{j-1}{N}} \frac{dL_1}{dW}(W_e) \left[W_e^\top W_e\right]^{\frac{N-j}{N}}\right) = \sum_{j=1}^{N} \left(\left[W_e W_e\right]^{\frac{N-j}{N}} \circ \left[W_e^\top W_e\right]^{\frac{j-1}{N}}\right) \cdot \mathrm{vec}\left(\frac{dL_1}{dW}(W_e)\right) \tag{25}$$

Therefore, the end-to-end update rule in Equation 24 can be rewritten as the standard gradient descent in the follwoing way:

$$\mathrm{vec}(W_e^{(t+1)}) = (1 - \eta\lambda N) \cdot \mathrm{vec}(W_e^{(t)}) - \eta \cdot P_{W_e^{(t)}} \cdot \mathrm{vec}\left(\frac{\partial L_1}{\partial W_e}(W_e^{(t)})\right), \tag{26}$$

where $P_{W_e^{(t)}}$ is a positive semi-definite preconditioning matrix that depends on $W_e^{(t)}$. If the singular values of $W_e^{(t)} \in \mathbb{R}^{k \times d}$ are denoted by $\sigma_1, \ldots, \sigma_{\max\{k,d\}}$ (with $\sigma_r = 0$ if $r > \min\{k, d\}$), and the corresponding left and right singular vectors by $u_1, \ldots, u_k \in \mathbb{R}^k$ and $v_1, \ldots, v_d \in \mathbb{R}^d$ respectively, the eigenvectors of $P_{W_e^{(t)}}$ are given by $\mathrm{vec}(u_r v_{r'}^\top)$ for $r = 1, \ldots, k$ and $r' = 1, \ldots, d$, with eigenvalues $\sum_{j=1}^{N} \sigma_r^{2(\frac{N-j}{N})} \sigma_{r'}^{2(\frac{j-1}{N})}$, $\quad r = 1, \ldots, k, \quad r' = 1, \ldots, d.$. So the transformation to the gradient in equation 27 can be seen as a preconditioning that depend on the singular value decomposition of W_e. For $N \geq 2$, an increase in the singular values of W_e (σ_r or $\sigma_{r'}$) leads to an increase in the eigenvalue corresponding to the eigen direction $u_r v_{r'}^\top$. This implies that the preconditioning favors directions that correspond to singular vectors whose presence in W_e is stronger. The study thus concludes that the effect of overparameterization, boils down to modifying gradient descent by promoting movement along directions that fall in line with the current location in parameter space. Since a common practice in deep learning is to initialize the weight near zero, the current location can be interpreted as the overall movement made and hence can be considered as a form of implicit acceleration. A key implication from the study is that the equations 24 and 26 depend on N which is the number of layers in the network but they do not depend on the hidden widths $n_1 \ldots n_{N-1}$. So overparameterizing wide or narrow networks have the same effect and it is the depth that matters. To further illustrate these acceleration effects, the study considered the case of single output where $k = 1$. Utilizing equation 26 we get the following result for the single output case:

$$W_e^{(t+1)} \leftarrow (1 - \eta\lambda N) \cdot W_e^{(t)} - \eta \left\|W_e^{(t)}\right\|_2^{2-\frac{2}{N}} \cdot \left(\frac{dL_1}{dW}(W_e^{(t)}) + (N-1) \cdot \mathrm{Pr}_{W_e^{(t)}}\left\{\frac{dL_1}{dW}(W_e^{(t)})\right\}\right) \tag{27}$$

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

$$\Pr_W\{V\} := \begin{cases} \frac{WW^\top}{\|W\|_2^2} \cdot V & \text{if } W \neq 0 \\ 0 & \text{if } W = 0 \end{cases} \tag{28}$$

So the effect of overparameterization on gradient descent can be interpreted as a combination of adaptive learning rate and an amplification in the direction of W_e. Due to this, these effects bear potential to accelerate the optimization and were utilized to obtain the experimental results.

6 Relevance to Regularization

Incorporating a regularizer into the objective function is a conventional technique to enhance optimization, although nowadays it is more commonly linked with generalization. The study proved that, in cases with a single output, the advantages of overparameterization cannot be replicated by simply adding a regularization term to the original training loss or through any comparable adjustments. This conclusion isn't immediately apparent since, unlike many acceleration methods that explicitly retain a memory of past gradients, updates in overparameterization inherently involve derivatives of the model parameters. The regularization term is defined as the following for the single output case from equation 27:

$$F_\phi(W) = \begin{cases} \|W\|_2^{2-\frac{2}{N}} \left(\phi(W) + (N-1)\frac{\langle \phi(W), \frac{W}{\|W\|}\rangle W}{\|W\|}\right), & \text{if } W \neq 0 \\ 0, & \text{if } W = 0 \end{cases}$$

Utilizing the fundamental theorem in line integrals [14], the study proved that there exists a positive lower bound for the line integral of $F(\cdot)$ which contradicts the statement of the fundamental theorem of line integrals that the integral of ∇g for any differentiable function g amounts to 0 along any closed curve.

7 Illustration of Acceleration

To understand the implicit acceleration due to overparameterization, consider linear regression, where vectors in $\mathbb{R}^2$ are assigned labels in $\mathbb{R}$. Suppose 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 that the loss function of interest is L_p, where $p \in 2\mathbb{N}$, defined as:$L_p(\hat{y}, y) = \frac{1}{p}(\hat{y}-y)^p$. The overall training loss can be expressed in terms of $w = [w_1, w_2]^\top$ as:

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

With a fixed learning rate $\eta > 0$ (and zero weight decay), gradient descent on $L(\cdot)$ updates as:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta(w^{(t)} - y_i)^{p-1}, \quad i = 1, 2.$$

Changing variables, introduce $\Delta_i = w_i - y_i$, then:

$$\Delta^{(t+1)} \leftarrow \Delta^{(t)} - \eta(\Delta^{(t)})^{p-2}, \quad i = 1, 2.$$

Assuming near zero initialization for the original weights w_1 and w_2, Δ_1 and Δ_2 start 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, that is given by:

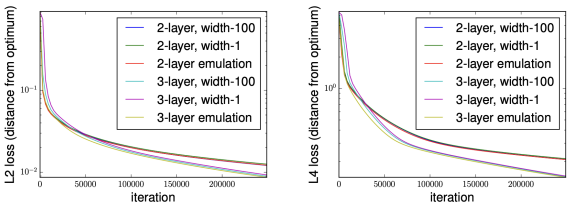
$$\eta < \frac{2}{y_i^{p-2}}, \quad i = 1, 2.$$

Consider an ill-conditioned problem with $y_1 \gg y_2$. If $p = 2$, this does not affect the bound for η [15]. However, if $p > 2$, the learning rate is constrained by y_1, causing Δ_2 to converge very slowly due to the lack of "communication" between the coordinates—a situation not exclusive to gradient descent but also common in other algorithms like AdaGrad, Adam, etc. Now, optimizing $L(\cdot)$ via overparameterization (with the update rule in Equation 26), the coordinates are coupled, and as Δ_1 decreases, the learning rate effectively scales by $\frac{2}{y_1^{2-\frac{2}{N}}}$. This allows for potentially faster convergence of Δ_2, thus providing the luxury to temporarily slow down Δ_2 to ensure that Δ_1 does not diverge, with the latter speeding up the former as it stabilizes, thus accelerating the overall convergence. The study further proves that while the standard gradient descent limits w_2 with a learning rate η^{GD}at almost $\frac{2}{y_1^2}$, where as the overparameterization adjusts w_2 with a learning rate $\eta^{OP} = \frac{1}{2\epsilon_1 y_1}$ for the special case of l_4 loss with $N = 2$ with the $y_1 \gg 1$ and $\epsilon_1 \ll 1$. The proof involved using the equation 26 for the specific case and then finding out the values of the parameters in the next iteration.

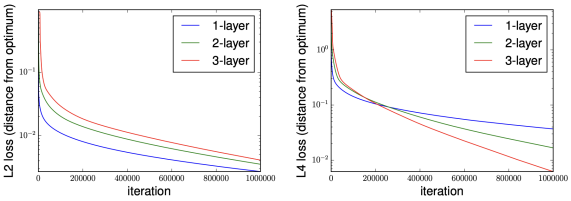
8 Experimental Results

The preliminary experiments evaluated the derived end to end update rule in Equation 24 on whether it is applicable to practical scenarios utilizing TensorFlow and focused on a scalar regression task from the UCI Machine Learning Repository's "Gas Sensor Array Drift at Different Concentrations" dataset. The results, displayed in Figure 1a, show that deeper networks do not necessarily offer faster convergence with l_2 loss but do exhibit improved performance with l_4 loss, suggesting depth-induced acceleration without explicit algorithmic enhancements. It also demonstrates the negligible effect of network width on convergence, consistent with the analysis. The learning rates in Figure 1a were not set optimally and so Figure 1b display the experiments for both l_2 loss and l_4 loss with learning rate chosen via grid search that gave the fastest convergence. The depth significantly accelerated convergence in the case l_4 loss.

Figure 1



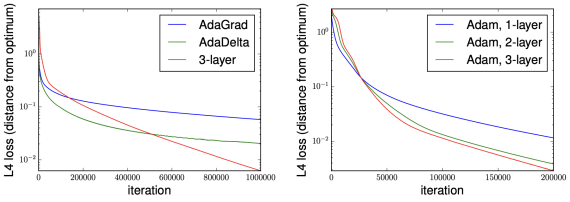
(a) GD vs Derived Preconditioning for L2 and L4 loss



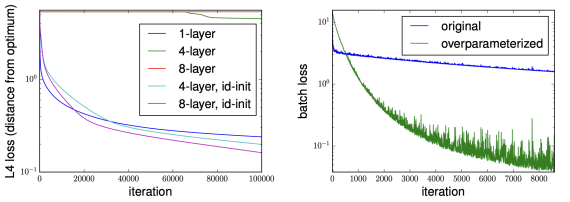
(b) GD vs Derived Preconditioning with optimal η

Figure 2a left shows the result of comparing the implicit acceleration of depth against explicit methods AdaGrad and AdaDelta with single layer for acceleration and adaptive learning. Surprisingly, for the l_4 loss, overparameterizing, thereby turning a convex problem non-convex, was observed to enhance the optimization than methods designed specifically for the convex problems with an exception of Adam which was considerably faster also shown in Figure 2a right. However, overparameterization simultaneously with Adam showed further acceleration implying that not only standard gradient descent benefits from the depth but also for other commonly used algorithms. The vanishing gradient problem in deep networks limits the depth in neural networks. One way to overcome this problem is by using identity initialization [16] that leads to linear residual networks. Experiments depicted in Figure 2b left were conducted comparing the convergence with near zero and identity initialization. It was observed that the identity initialization showed immediate progress in convergence, demonstrating the practicality of overparameterization under specific conditions. Further investigations into the effect of overparameterization on optimization for non-idealized but simple convolutional networks on the MNIST dataset were made. The overparameterization was introduced by simply replacing two matrices in succession instead of the matrix in each dense layer. The results in Figure 2b right confirmed that overparameterization could significantly speed up optimization, even with only a minimal (15 percent) increase in the number of parameters. This series of experiments supports the thesis that overparameterization, even in the absence of

Figure 2



(a) Left: GD vs Adagrad, AdaDelta; Right: Adam



(b) Left: Different Initialization; Right: SGD for CNN

expressiveness gains, can facilitate optimization, although the effects can vary based on network configuration and initialization strategy. This suggests a need for continued exploration into how depth and configuration choices impact training dynamics in neural networks.

9 Conclusion

This study confirmed that enhancing a neural network's depth, thus overparameterizing it, can indeed accelerate the optimization process across various simple scenarios. A thorough examination of linear neural networks, which have been prominently featured in recent research, revealed an intriguing insight: overparameterization

serves as an effective preconditioning mechanism with a clear mathematical formulation. This effect merges the aspects of adaptive learning rates and momentum, depending predominantly on the depth rather than the width of the network. Expanding a full theoretical framework to encompass nonlinear networks remains a formidable challenge; however, the empirical findings indicate that straightforward alterations, like dividing a single weight matrix into two, can substantially improve optimization without compromising expressiveness, as evidenced in Figure 2-right. Notably, applying gradient descent to classical convex problems such as linear regression with L_p loss ($p > 2$) can gain from shifting to a non-convex, over-parameterized setup. This discovery challenges established beliefs and encourages additional exploration. Future studies should strive to precisely measure this phenomenon, possibly integrating it with well-known acceleration methods such as momentum or adaptive regularization (like AdaGrad). Such investigations might open new avenues for understanding and leveraging overparameterization in neural network training.

References

[1] Arora, S., Cohen, N., & Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. In A. Krause & J. Dy (Eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018* (Vol. 1, pp. 372-389). International Machine Learning Society (IMLS).

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

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

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

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

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

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

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

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

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

[11] 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.

[12] 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.

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

[14] Buck, R. C. Advanced calculus. Waveland Press, 2003.

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

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