Model Architecture Controls Gradient Descent Dynamics: A Combinatorial Path-Based Formula

Anonymous authors
Paper under double-blind review

Abstract

Recently, there has been a growing interest in automatically exploring neural network architecture design space with the goal of finding an architecture that improves performance (characterized as improved accuracy, speed of training, or resource requirements). However, our theoretical understanding of how model architecture affects performance or accuracy is limited. In this paper, we study the impact of model architecture on the speed of training in the context of gradient descent optimization. We model gradient descent as a first-order ODE and use ODE's coefficient matrix H to characterize the convergence rate. We introduce a simple analysis technique that enumerates H in terms of all possible "paths" in the network. We show that changes in model architecture parameters reflect as changes in the number of paths and the properties of each path, which jointly control the speed of convergence. We believe our analysis technique is useful in reasoning about more complex model architecture modifications.

1 Introduction

Gradient descent and its variants are the cornerstones of deep learning. The theoretical properties of gradient descent have been widely studied in the literature; study of the convergence bounds and guarantees (5; 11; 13; 12), the characterization of the local geometry of stationary points (17; 10; 9; 16; 21; 26; 20; 19; 12), exploration of better algorithms to optimize the descent process (23; 14; 22; 27; 6) are just a few example of active research areas in this domain. Another major research area is the exploration of network architecture and its impact on performance. In recent years, network architecture search and design have shown major performance boosts in various aspects of deep learning (30; 29; 15; 7; 28).

Recent theoretical results are trying to relate model architecture and gradient descent properties. It has been shown that over-parametrization in width guarantees convergence in deep neural network (13). It has also been shown that increasing depth has an impact similar to adding momentum optimization and adaptive learning rate to the objective function (3).

The gradient descent process can be described by a system of first-order differential equations in the continuous-time limit, in the form of $\dot{\mathbf{x}}(t) = \mathbf{H}(t)\mathbf{x}(t)$. In this paper, we study the properties of the coefficient matrix $\mathbf{H}(t)$, which governs the dynamics of the gradient descent. Specifically, we formulate $\mathbf{H}(t)$ in terms of all possible paths in the network. This representation is powerful as it enables us to analyze the gradient behavior symbolically and abstracts away the unnecessary bookkeeping of derivatives and weights. Specifically, we use this representation to explore the followings. (1) We study the impact of width on convergence. Using a simple path counting argument, we show that $\mathbf{H}(t)$ is a sum of m i.i.d. terms where m is dictated by the width of the network. This has been implied in the work of Simon Du, et al. (11) for the special case of a 2-layer RELU-activated network. We contrast the ease of arriving at such conclusions using our representation against methods used in previous work (11; 13; 12; 5). This also answers the open question of why a 2-layer student network needs to be wider than the teacher network to train fast. (2) We study the impact of depth on convergence. We show that adding a new layer leads to $\mathbf{H}(t)$ being decomposed into an adaptive learning rate term and a momentum term. Previous work (2) has observed this for a special case of fully connected linear neural networks.

2 Related Work

Related work can be categorized into three groups which we will explain in detail below. In general, the common theme across previous works is to show that if some conditions, such as bound on models' width or depth or assumptions on geometrical properties of the loss landscape are met, gradient descent or its variance converges to a global minimum.

Landscape Analysis Researchers have extensively studied the properties of error surface in linear neural network ((9; 17)) and have proved convergence to a global minimum for non-convex optimizations if error surface shows certain properties, such as all saddle points to be strict (i.e. there exists a negative curvature)((10; 16; 21; 26; 20; 19; 12)). However, it has been shown that the strict saddle property is not guaranteed for deep neural networks.

Width Analysis A number of recent work have characterized minimum width that guarantees convergence in terms of model architecture parameters and input data. We on the other hand study how convergence rate varies with width. Examples of finding width bounds include (13) where it has been shown that for a 2-layer ReLU-activated neural network with squared loss, as long as every hidden layer is wide enough, gradient descent converges to a global minimum at a linear rate. In (25), it is extended to fully connected linear networks and ResNets. Du et. al. (11) has also generalized this work to an L-layer fully connected linear neural network and shows the convergence rate as a function of depth, output dimension and least eigenvalue of the Gram matrix for large enough hidden layers. Ardalani et. al. (1) have shown empirically that for a wide range of RNN applications, increasing width will reduce the number of steps to minimum validation loss. Neural tangent kernel (18) studies convergence in the infinite width limit.

Depth Analysis Bartlett et al. (5) have proven for a deep linear neural network with isotropic input and identity initialization that the number of steps to ϵ-proximity of the best answer is polynomial in the number of layers. Others (2; 3) have shown that over-parametrization introduced by depth can accelerate training under assumptions on initialization (balanced) and input data being whitened. Neural ODE is a study in the infinite depth limit (8).

3 Notations and Conventions

Matrix entries are denoted with row and column indices, for instance $H = (H_{ij})$ is a matrix with entries H_{ij}. When H is diagonalizable, we denote by $\lambda_{min}(H)$ its smallest eigenvalue. For a matrix W, $W(i,:)$ and $W(:,j)$ denote the row vector in the i-th row and the column vector in the j-th column, respectively. Write $\sum_{w\in W}$ for summing over all elements of W. We use pairs $\{X_i, y_i\}, 1 \le i \le N$ to represent labeled data where X_i is input and y_i is the output. We write $X_{i,k}$ for the k-th component of X_i. We denote loss by L. We work with feed-forward neural networks, which we sometimes denote abstractly by a function f, or $f(w,x)$ as a function of input x and weights w. We denote network weights individually by w_i. Predictions are denoted by $u_i = f(w, X_i)$ $1 \le i \le N$. The prediction on all inputs is denoted by the vector $\mathbf{u}$. l_p loss is defined as $\sum(y-u_i)^p$ for even integers $p \ge 2$. We use angle brackets to denote inner products, i.e. $\langle \overrightarrow{v}, \overrightarrow{w} \rangle = \sum_i v_i w_i$ represents inner product of two vectors. i.i.d. random variables stands for independent and identically distributed random variables. ODE stands for ordinary differential equations. NN stands for neural networks. GD stands for gradient descent.

4 Main Formula

4.1 Preliminaries

Convergence rate can be characterized as the rate of change in loss during the training process. If steps are infinitesimally small, the rate of change in loss can be characterized as:

$$\frac{dL}{dt} = \sum_{i=1}^{N} \frac{\partial L}{\partial u_i} \cdot \frac{du_i}{dt} \tag{1}$$

Where $u_i(t) = f(w(t), x_i)$ is the prediction on input x_i at time t. For simplicity we focus on the dynamics of $\frac{\mathbf{du_i}}{\mathbf{dt}}$, which can be expanded as follows:

$$\frac{du_i}{dt} = \sum_{r=1}^{m} \frac{\partial u_i}{\partial w_r} \cdot \frac{dw_r}{dt} = \sum_{r=1}^{m} \big(\frac{\partial u_i}{\partial w_r} \sum_{j=1}^{N} -\eta \frac{\partial L}{\partial u_j} \cdot \frac{\partial u_j}{\partial w_r}\big)$$
$$= -\eta \sum_{j=1}^{N} \sum_{r=1}^{m} \big(\frac{\partial u_i}{\partial w_r} \frac{\partial u_j}{\partial w_r} \cdot \frac{\partial L}{\partial u_j}\big) = -\eta \sum_{j=1}^{N} \big(\sum_{r=1}^{m} \frac{\partial u_i}{\partial w_r} \frac{\partial u_j}{\partial w_r}\big) \cdot \frac{\partial L}{\partial u_j} \tag{2}$$

We use the definition of gradient descent and chain rule to derive the equation above. Gradient descent at infinitesimal small step is defined by:

$$\frac{dw_i}{dt} = -\eta \frac{\partial L}{\partial w_i} = -\eta \sum_{j=1}^{N} \frac{\partial L}{\partial u_j} \cdot \frac{\partial u_j}{\partial w_i} \tag{3}$$

Observe that $\overrightarrow{\frac{\partial L}{\partial u}}$ is a column vector. Hence, if we define H_{ij} as:

$$H_{ij} = \sum_{r=1}^{m} \frac{\partial u_i}{\partial w_r} \frac{\partial u_j}{\partial w_r} \tag{4}$$

Then we can simplify equation (2) in a vector form as:

$$\begin{pmatrix} \frac{du_1}{dt} \\ \vdots \\ \frac{du_N}{dt} \end{pmatrix} = \eta H \cdot \begin{pmatrix} \frac{\partial L}{\partial u_1} \\ \vdots \\ \frac{\partial L}{\partial u_N} \end{pmatrix} \tag{5}$$

Assuming L is l_2 loss, equation (5) simplifies to the following. This result will hold for other l_p losses (see Appendix A).

$$\frac{d\mathbf{u}}{dt} = \eta \mathbf{H}(\mathbf{y} - \mathbf{u}) \tag{6}$$

$$\frac{d(\mathbf{y} - \mathbf{u})}{dt} = -\frac{d\mathbf{u}}{dt} = -\eta \mathbf{H} \cdot (\mathbf{y} - \mathbf{u}) \tag{7}$$

Notice that this puts $\mathbf{y} - \mathbf{u}$ in a system of differential equations whose single variable analogue is:

$$\frac{df(t)}{dt} = hf(t) \tag{8}$$

The above equation's solution is:

$$f(t) = c_0 e^{ht} = c_0 (1 - \xi)^t \tag{9}$$

when $h < 0$, $e^h < 1$ and let $e^h = 1 - \xi$. ξ governs the convergence rate in Equation (9). In the actual system of equations (7), ξ will be determined by the minimum eigenvalue of $H(t)$, $\lambda_{min}(H(t)))$ as we will explain next.

4.2 Relationship between $\lambda_{min}(H(t))$ and Convergence

In this subsection, we briefly explain why $\lambda_{min}(H(t))$ replaces the ODE coefficient h as convergence rate. For more insight we refer the readers to previous literature such as (13). By rewriting equation (6) in a discrete space, we have:

$$u(k+1) - u(k) = \eta H(k)(y - u(k)) \implies -u(k+1) = -\eta H(k) y - u(k) + \eta H(k) u(k) \tag{10}$$

Then:

$$y - u(k+1) = (1 - \eta H(k))(y - u(k)) = (1 - \eta H(k))(1 - \eta H(k-1))(y - u(k-1)) = \dots \tag{11}$$

Hence, suppose $\lambda_0 = \min_k(\lambda_{min}(H(k))) > 0$, we have a convergence rate as follows:

$$||y - u(k)|| \leq (1 - \eta \lambda_0)^k ||y - u(0)|| \tag{12}$$

This implies an exponential decay, also referred to as "linear" convergence in the literature.

4.3 CHARACTERIZING H IN TERMS OF NETWORK PATHS

In order to state the path decomposition formula, we first establish some notations.

- We denote the neural network abstractly with a function $f(X)$.
- For any neural network, we view the underlying network as a **directed graph** where edges are in the backward propagation direction.
- We define **path** p, as a series of connected nodes and directed edges. We define **an output path** as a path that starts with the output node.
- We **index** the nodes as follows. Index the output node by 0, the one after by 1, and so on. We denote the j-th node in the i-th path by p^i_j.
- We allow different activation functions at nodes. We denote the **activation function** at p_s by σ_s.
- An example path that ends in an input node may look like this:

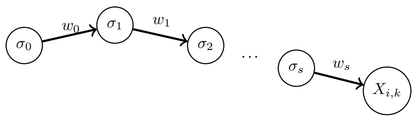


Figure 1: Illustration of a multilayer perceptron

- The **post-subnetwork at $\mathbf{w_s}$** is defined as the network formed by the subgraph rooted at the right vertex of w_s. The **pre-subnetwork at $\mathbf{w_s}$** is defined as the union of all output paths ending in w_s, denoted by $pre(w_s)$.
- For every input X_i, we denote the **activation** value at node w_s with $\mathrm{activation}(w_s, X_i)$ or $\mathbf{f[w_s](X_i)}$. Note that this is equivalent to the value at the right vertex of w_s once X_i propagated through the post-subnetwork at w_s.
- We denote the local gradient wrt. input at node s along the path p with $\tau^i_{p_s}$.
- Given the pre-subnetwork at $\mathbf{w_s}$, we write the explicit formula for $\frac{\partial u_i}{\partial w_s}$:

$$\frac{\partial u_i}{\partial w_s} = \mathrm{activation}(w_s, X_i) \cdot \sum_{p \in pre(w_s)} \mathrm{path\_gradient_p}(w_s, X_i) \tag{13}$$

  where $\mathrm{path\_gradient}_p(w_s, X_i)$ is the gradient value along the path p at the left vertex of w_s, and can be expanded as follows using the chain rule:

$$\mathrm{path\_gradient}_p(w_s, X_i) = (\tau^i_0 w_0 \tau^i_{p_1} w_1 \dots \tau^i_{p_{l(p)-1}} w_{l(p)-1})(p) \tag{14}$$

  we refer to $\mathrm{path\_gradient}_p(w_s, X_i)$ as $\mathrm{PG}_p(w_s, X_i)$ for brevity from now on.

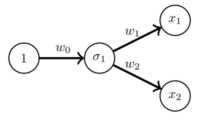
Theorem 4.1. *For a network f,*

$$H_{ij} = \sum_w \left(f[w](X_i) \cdot f[w](X_j) \cdot \Big(\sum_{p \in pre(w)} \mathrm{PG}_p(w, X_i) \Big) \cdot \Big(\sum_{p \in pre(w)} \mathrm{PG}_p(w, X_j) \Big) \right) \tag{15}$$

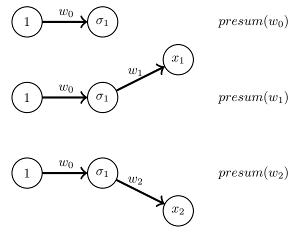
which easily follows by substituting Equation (13) in Equation (4). In the rest of the paper, we will make many arguments most of which rely on the number of terms in this equation. Number of terms in Equation 15 is itself controlled by the number of paths in the pre-subnetworks at each weight. Following examples provide some intuition.

Example 4.2. *In this example, our network has a structure as illustrated in Figure 2a. The output node is denoted with value one, indicating that the activation function at output layer is an identity function. As shown in Figure 2b, there are only one path in each weight's pre-subnetwork. Therefore, there are only three terms in Equation 15, where each summand is of form $\mathrm{PG}_p(w, X_i)\mathrm{PG}_p(w, X_j)$. We refer to these terms as symmetrical pairs since both multipliers are representing the same path.*

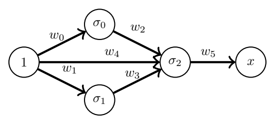
Example 4.3. *A more complicated application of the formula can be found in Figure 2d. There is one path in each of the following weights' presubnetwork: w_0,w_1,w_2,w_3, and w_4, and three paths at w_5's presubnetwork. Therefore, Equation 15 would have $5 + 3^2 = 14$ summands, 8 of which are terms with symmetrical pairs and 6 are cross-terms induced by the paths in w_5's pre-subnetwork. In general, cross-terms will appear only if there are two paths that share a final edge which can only happen if there are three or more number of layers in the network.*



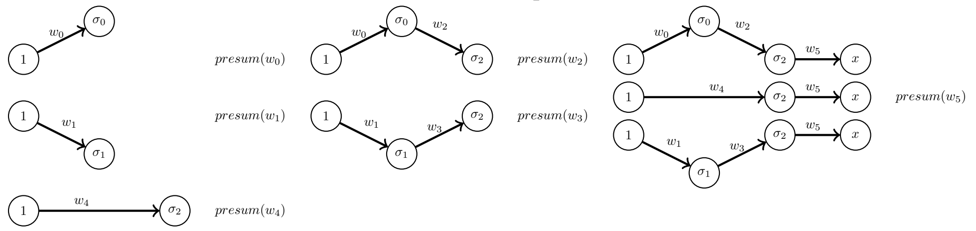
(a) Example 4.2



(b) Paths Example 4.2



(c) Example 4.3



(d) Paths Example 4.3

Figure 2: Path Decomposition Examples.

5 Architecture Implications

5.1 Convergence Rate Scales Linearly with Width

In this Section, we show that for sufficiently wide neural networks, convergence rate has a linear relationship with the network's width. Results are drawn theoretically at time 0 assuming random i.i.d initialization. For these results to generalize across time, we assume that H stays close to its initial value. In the Appendices D and E, we show empirical results that suggest this assumption is true. A heuristic justification for the persistence of these properties across training time is that over-parameterization allow weigh vectors to vary small amounts to change loss but overall stay close to their initialization for all iterations. For the extreme case of infinitely-wide networks, it has been shown that H is constant throughout training (18). Others (4; 13) have also shown that for sufficiently-wide neural networks weights remain close to initialization, and so our argument holds true across training.

Proposition 5.1. *For a 2-layer neural network with hidden dimension m, the convergence rate scales linearly with m.*

To demonstrate the simplicity and expressiveness of our approach, next we will discuss proofs with and without our approach.

Brute-force approach A 2-layer fully-connected linear neural network can be modeled as:

$$u(X) = V \times W \times X = (v_1 \quad ... \quad v_m) \times \begin{pmatrix} w_{11} & ... & w_{1d} \\ \vdots & & \vdots \\ w_{m1} & ... & w_{md} \end{pmatrix} \times \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \tag{16}$$

where $V \in \mathbb{R}^{1,m}$ and $W \in \mathbb{R}^{m,d}$ are the weights of the second and first layer, respectively and the input X is a d-dimensional column vector. We begin the analysis with the simple case of a 2-layer linear network where the network's width is one, i.e. $m = 1$ and $u(X) = v_0 \cdot \langle W(:, 1), X \rangle$.

H_{ij} can be decomposed into two terms, partial derivatives with respect to weights of the first layer and partial derivatives with respect to weights of the second layer.

$$H_{ij} = \frac{\partial u_i}{\partial v_0}\frac{\partial u_j}{\partial v_0} + \sum_{w \in W} \frac{\partial u_i}{\partial w}\frac{\partial u_j}{\partial w} = \langle W(:, 1), X_i \rangle \cdot \langle W(:, 1), X_j \rangle + v_0^2 \sum_{k=1}^{d} X_{i,k} \cdot X_{j,k} \tag{17}$$

For a more general case where the width is m, we have:

$$H_{ij} = \sum_{l=1}^{m} \left(\langle W(:, l), X_i \rangle \cdot \langle W(:, l), X_j \rangle + v_l^2 \sum_{k=1}^{d} X_{i,k} \cdot X_{j,k} \right) \tag{18}$$

Adding non-linearity would make this equation even more complex, which is the approach used in previous literature together with induction:

$$H_{ij} = \sum_{l=1}^{m} \left(\sigma(\langle W(:,l), X_i \rangle) \cdot \sigma(\langle W(:,l), X_j \rangle) + v_l^2 \sum_{k=1}^{d} X_{i,k} \cdot \sigma'(\langle W(:,l), X_i \rangle) \cdot X_{j,k} \cdot \sigma'(\langle W(:,l), X_j \rangle) \right) \tag{19}$$

See Appendix B for details of how we derive these equations. Since we assume all weights (w_{lk}, v_0) are independent and identically distributed, the summands within Equations (18) and (19) are also independent and identically distributed. This indicates that H_{ij} is composed of m i.i.d components, hence convergence rate scales linearly with m.

Our approach: Using Theorem 4.1 More generally, by applying Theorem 4.1, it is immediate to see every pre-subnetwork will be a path in a 2-layer fully connected network. There are two types of paths; m length 1 paths and m length 2 paths. The weights are independent and follow the same distribution, hence, a 2-layer network's H matrix always decomposes as m i.i.d sums. Hence convergence rate scales linearly with m. This proves Proposition 5.1. □

See Section 6 for empirical analysis that motivated this study. We show for a 2-layer linear and non-linear networks that convergence rate has linear relationship with width.

Remark 5.2 (Properties of H^∞)**.** *Others (13) have studied the properties of $H^\infty = \frac{1}{m} \cdot H$ and have shown that it is constant at infinite width limit. Using Prop. 5.1, it is easy to see that $\frac{1}{m} \cdot H$ converges to a constant value as m approaches infinity.*

5.2 Depth-induced Momentum

For a special case of linear neural networks, it has been shown that overparametrization through depth results in implicit acceleration in training (2; 3). Specifically, it has been shown that impact is much similar to the momentum acceleration (24), hence the term depth-induced momentum. Here, we will showcase how a similar conclusion is derivable using our explicit path formula. Take a network f. Construct a new network g by adding a multiplication layer on top, i.e. $g(X) = \mu f(X)$ with an extra trainable parameter μ. Then in g, there's a new path of length 1 whose edge is μ and all paths in f are annexed by μ at the front. Hence H_{ij} for the new network g can be computed in terms of H_{ij} for the network f as follows:

$$H_{ij}(g) = \mu^2 H_{ij}(f) + f(X_i) \cdot f(X_j) \tag{20}$$

This can be written in the matrix form as:

$$H(g) = \mu^2 H(f) + K \tag{21}$$

where K is the matrix whose elements at K_{ij} are: $f_i f_j = \frac{1}{\mu^2} g_i . g_j$. Note here that K is a symmetric positive semidefinite matrix with only one nonzero eigenvalue $\frac{1}{\mu^2}(g_1^2 + \cdots + g_n^2)$ with associated eigvenvector $(g_1, ..., g_N)$ (Proof in Appendix F). From Equation 5.2, it is easy to see that eigenvalues of $H(g)$ are pushed in the direction of g. So the updates $\frac{du_1}{dt}, ... \frac{du}{dt}$ prefer the direction of $u_1, ...u_n$ which itself carries accumulative information about the past directions. This is similar to the concept of momentum acceleration where the gradient dynamics from the past controls the future direction.

With our formula, the argument applies to the non-linear network without change, as we are not making any assumptions about the structure of network f or activation functions. See Experiment Depth induced momentum.

5.3 On the Importance of Number of Paths for Convergence

Here, we will provide some heuristic arguments, based on our formula, on why over-parametrization through increasing the number of paths can be more effective than increasing the number of nodes (parameters) and conclude that increasing depth is more effective than increasing width for accelerating the training process. We first introduce an important terminology which we will use below.

Definition 5.3. *Given a collection of vectors $\{g_i\}$, the Gram matrix G is defined as a matrix whose ij-th entry is $\langle g_i, g_j \rangle$ under the Euclidean inner product.*

Remark 5.4. *Based on the definition above it is easy to see that Gram matrix is symmetric and positive-semidefinite and its rank is exactly the dimension of the subspace spanned by $\{g_i\}$. Suppose*

$H = H_1 + \cdots + H_n$ are all Gram matrices. Then H is positive definite if any of the H_i is positive definite. This is because of supperadditivity of the minimum eigenvalue function, i.e. $\lambda_{min}(H) \geq \lambda_{min}(H_1) + \cdots + \lambda_{min}(H_n)$.

Strongly-Gram vs. Weakly-Gram Equation(15) in Theorem 4.1 can be expanded further as follows and the terms can be classified into two groups. The first group will include all the symmetrical pairs and the second group will include all the cross-terms.

$$\begin{aligned}
H_{ij}(f) &= \sum_w \left(\sum_{p \in pre(w)} \mathrm{PG}_p(w, X_i) f[w](X_i) \right) \cdot \left(\sum_{p \in pre(w_s)} \mathrm{PG}_p(w, `X_j) f[w](X_j) \right) \\
&= \sum_w \sum_{p \in pre(w)} \mathrm{PG}_p(w, X_i) f[w](X_i) \cdot \mathrm{PG}_p(w, X_j) f[w](X_j) + \qquad (22) \\
&\sum_w \sum_{\substack{p_1, p_2 \in pre(w) \\ p_1 \neq p_2}} \mathrm{PG}_{p_1}(w, X_i) f[w](X_i) \cdot \mathrm{PG}_{p_2}(w, X_j) f[w](X_j) \\
&= \sum_p \mathrm{PG}_p(w, X_i) f[w](X_i) \cdot \mathrm{PG}_p(w, X_j) f[w](X_j) + \\
&\sum_w \sum_{\substack{p_1, p_2 \in pre(w) \\ p_1 \neq p_2}} \mathrm{PG}_{p_1}(w, X_i) f[w](X_i) \cdot \mathrm{PG}_{p_2}(w, X_j) f[w](X_j) \\
&= H^S_{ij}(f) + H^W_{ij}(f) \qquad (23)
\end{aligned}$$

We refer to the first group as **strongly Gram**, and denote it by $H^S_{ij}(f)$, and refer to the second group as **weakly Gram** and denote it by $H^W_{ij}(f)$, i.e. we have:

Remark 5.5 (Gramness of the strongly-Gram component). *Suppose there are q paths in the network, $p_1, p_2, \ldots, p_q$. Then Equation (22) implies that the strongly Gram component is Gramian with vectors*

$$g_i = \left(\mathrm{PG}_{p_1}(w, X_i) f[w](X_i), \quad \ldots \quad , \mathrm{PG}_{p_q}(w, X_i) f[w](X_i) \right) \qquad (24)$$

Remark 5.6 (Insignificance of the weakly Gram component). *We claim that the weakly Gram component is insignificant, see Appendix C for heuristics and empirical evidence.*

Remark 5.7 (Number paths vs. number of nodes). *The Gram matrix H^S should ideally be of full rank for good convergence. So we want the subspace spanned by $\{g_i\}$ to have dimension at least N. N is the dimension of H and also the number of sample points. On the other hand, as shown in Equation 24 g_i's are embedded as vectors in a q-dimensional vector space where q is the number of paths.Therefore, to ensure H is full-rank, q should satisfy $q \geq N$. The bigger q is, the more likely they are linearly independent and make the matrix full rank. In deep learning, N is usually large, therefore, we need to have networks whose number of paths is comparable to N for best convergence.*

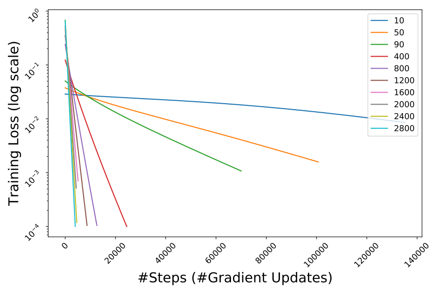
Remark 5.8 (Depth vs. width). *Adding a new parameter at a new layer can increase the number of paths more effectively than adding it at an existing layer. In other words, making models deeper increases the number of paths faster than making models wider.*

6 Experiments

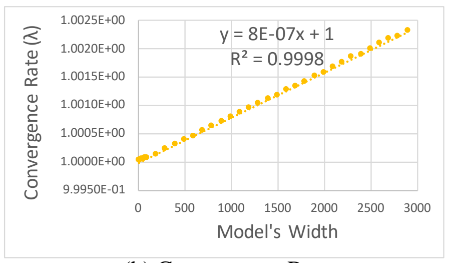
In this section, we provide experimental results that demonstrate the impact of model structure on convergence.

Setup We employ a teacher-student framework to put our ideas to test. Our teacher model is a two-layer neural network with hidden dimension 100, input dimension 50 and output dimension 1. We assume a Gaussian distribution both for the inputs and ground-truth weights. The student models are also two-layer neural networks with variable hidden dimensions, and l_2 loss. We assume the training curve follows the $y_{(k)} = \lambda^{-k} y_{(0)}$ equation, where $y_{(k)}$ is the loss at k_{th} step, and λ is the rate of convergence. We study how λ vary with model size.

Two-Layer Linear Neural Network As shown in Figure 3a, training curve has a power law relationship with number of steps which can be perfectly characterized by the $y_{(k)} = \lambda^{-k} y_{(0)}$ equation. As shown in Figure 3b, convergence rate (λ) grows linearly with width.

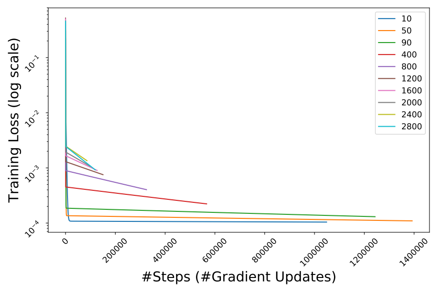


(a) Training Curve

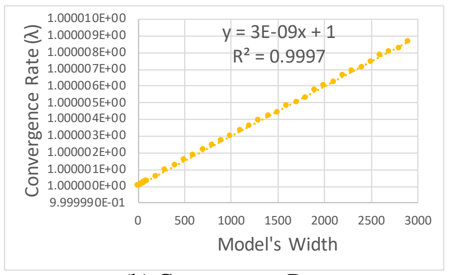


(b) Convergence Rate

Figure 3: **Training Curve Characterization for Two-layer Linear Neural Network.**



(a) Training Curve



(b) Convergence Rate

Figure 4: **Training Curve Characterization for Two-layer Non-Linear Neural Network.**

Two-Layer Non-Linear Neural Network As shown in Figure 4a, training curve for models with the sigmoid non-linearity begins in a steep region where loss drops very quickly for a short period of time and ends in a steady region where loss drops very slowly. We are interested in characterizing the curve in the steady region. As shown in Figure 4b, convergence rate (λ) grows linearly with width.

Depth induced momentum Figure 5 compares the convergence rate curves for 2-layer, 3-layer and 4-layer NN using GD, against a 2-layer network with a momentum optimizer. The deeper networks are constructed from 2-layer networks with additional scalar multiplication layers. We quantify the convergence rate at step t as $1 - \frac{Loss_{t+1}}{Loss_t}$. As discusses in Section 4.5, one can observe depth-induced momentum in convergence rates, similar to using momentum optimizer with a 2-layer NN.

7 Conclusion

In this paper, we provide a path-based combinatorial formula for the coefficient matrix $\mathbf{H}$ which governs the gradient descent dynamics. We use this representation to explore the impact of network structure on convergence symbolically. We believe this high-level representation is simple to work with, is more generalizable and interpretable than previous work; we were able to explain how width and depth affect gradient descent convergence rate with simple reasoning about the number of paths in the network.

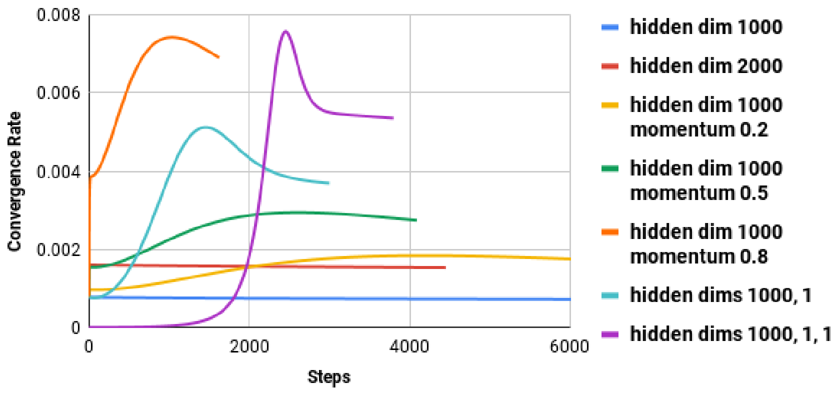


Figure 5: **Depth induced momentum.**

REFERENCES

[1] Joel Hestness Ardalani, Newsha and Gregory Diamos. Have a larger cake and eat it faster too: A guideline to train larger models faster. *http://research.baidu.com/Public/uploads/5abdde96f0ca3.pdf*, 2018.

[2] Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018.

[3] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018.

[4] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

[5] Peter L Bartlett, David P Helmbold, and Philip M Long. Gradient descent with identity initialization efficiently learns positive-definite linear transformations by deep residual networks. *Neural computation*, 31(3):477–502, 2019.

[6] Raef Bassily, Mikhail Belkin, and Siyuan Ma. On exponential convergence of sgd in non-convex over-parametrized learning. *arXiv preprint arXiv:1811.02564*, 2018.

[7] Lingjiao Chen, Hongyi Wang, Jinman Zhao, Dimitris Papailiopoulos, and Paraschos Koutris. The effect of network width on the performance of large-batch training. In *Advances in Neural Information Processing Systems*, pages 9302–9309, 2018.

[8] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[9] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[10] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[11] Simon S Du and Wei Hu. Width provably matters in optimization for deep linear neural networks. *arXiv preprint arXiv:1901.08572*, 2019.

[12] Simon S Du, Chi Jin, Jason D Lee, Michael I Jordan, Aarti Singh, and Barnabas Poczos. Gradient descent can take exponential time to escape saddle points. In *Advances in neural information processing systems*, pages 1067–1077, 2017.

[13] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

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

[15] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[16] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

[17] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[18] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[19] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1724–1732. JMLR. org, 2017.

[20] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv preprint arXiv:1609.04836*, 2016.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Yu Nesterov. Introductory lectures on convex programming. 1998.

[24] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[25] Haochuan Li Liwei Wang Xiyu Zhai Simon S. Du, Jason D. Lee. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[26] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[27] Yazhen Wang. Asymptotic analysis via stochastic differential equations of gradient descent algorithms in statistical and computational paradigms. *arXiv preprint arXiv:1711.09514*, 2017.

[28] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[29] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

[30] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

A l_p LOSS

We argue very briefly that without loss of generality, we can assume that we work with l_2 loss instead of other L_p norms. For l_p loss,

$$\frac{\partial L}{\partial u_j} = p \cdot (y_j - u_j)^{p-1} \tag{25}$$

So for l_2 loss, Equation 5 becomes

$$\frac{d\mathbf{u}}{dt} = \eta \mathbf{H}(\mathbf{y} - \mathbf{u}) \tag{26}$$

Recall the argument for decay rate under Equation 8. The single variable analogue of the system is:

$$\frac{df(t)}{dt} = hf(t) \tag{27}$$

the solution looks like:

$$f(t) = c_0 e^{ht} \tag{28}$$

and decays like:

$$f(t) = c_0(1-\xi)^t \tag{29}$$

with

$$\xi = 1 - e^h \tag{30}$$

When h is small, by Taylor expansion

$$\xi \approx 1 - (1 + h + \frac{h^2}{2}...) \approx -h \tag{31}$$

Instead of using the closed form solution, this ODE could equivalently be solved using the series method and we would be getting the Taylor expansion of equation (28) on the right and equation (31) remains the same. Note that the series method (more fundamentally, Taylor expansion involving only one matrix) works for both ODEs and systems with convergent assumptions on the coefficient matrix H. ηH is small by assumption, so we have convergent Taylor expansions, so we continue to argue for l_p loss as if we are in the single variable case.

For general l_p loss, we have:

$$f' = hf^{p-1} \tag{32}$$

then

$$f = c_0 e^{\frac{h}{p}t} \tag{33}$$

and the decay rate would be $\frac{h}{p}$, scaled down by p, so all the convergence rate statements we make for the rest of the paper for l_2 remains true if scaled by $\frac{p}{2}$.

B 2 LAYER LINEAR NETWORK

The detailed brute-force calculation of H_{ij} for a 2-layer fully connected linear network with $m = 1$ is as follows. Recall m is the width of the hidden layer.

$$\begin{aligned} H_{ij} &= \sum_{v \in V} \frac{\partial u_i}{\partial v_s}\frac{\partial u_j}{\partial v_s} + \sum_{w \in W} \frac{\partial u_i}{\partial w}\frac{\partial u_j}{\partial w} \\ &= \langle W(:,1), X_i \rangle \cdot \langle W(:,1), X_j \rangle + \sum_{k=1}^{d} (v_0 \cdot w_{1k} \cdot X_{i,k}) \cdot (v_0 \cdot w_{1k} \cdot X_{j,k}) \\ &= \langle W(:,1), X_i \rangle \cdot \langle W(:,1), X_j \rangle + v_0^2 \sum_{k=1}^{d} (w_{1k} \cdot X_{i,k}) \cdot (w_{1k} \cdot X_{j,k}) \end{aligned} \tag{34}$$

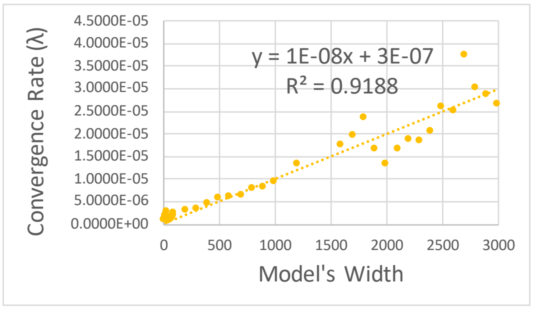


Figure 6: Convergence Rate for a 3-layer RELU-activated Network.

For a general m, we have:

$$\begin{aligned}
H_{ij} &= \sum_{v \in V} \frac{\partial u_i}{\partial v_s} \frac{\partial u_j}{\partial v_s} + \sum_{w \in W} \frac{\partial u_i}{\partial w} \frac{\partial u_j}{\partial w} \\
&= \sum_{l=1}^{m} \langle W(:,l), X_i \rangle \cdot \langle W(:,l), X_j \rangle + \sum_{l=1}^{m} \sum_{k=1}^{d} (v_l \cdot X_{i,k}) \cdot (v_l \cdot X_{j,k}) \\
&= \sum_{l=1}^{m} \left(\langle W(:,l), X_i \rangle \cdot \langle W(:,l), X_j \rangle + \sum_{k=1}^{d} (v_l \cdot X_{i,k}) \cdot (v_l \cdot X_{j,k}) \right) \\
&= \sum_{l=1}^{m} \left(\langle W(:,l), X_i \rangle \cdot \langle W(:,l), X_j \rangle + v_l^2 \sum_{k=1}^{d} X_{i,k} \cdot X_{j,k} \right) \qquad (35)
\end{aligned}$$

C Weakly Gram matrix is empirically insignificant

In terms of network architecture, the weakly Gram term would only appear when two paths share a final edge. That means the network needs to have at least 3 layers. The number of pairs could grow quadratically with width. Heuristically, each term $\mathbb{E}[\mathrm{PG}_{p_1}(w, X_i) \cdot \mathrm{PG}_{p_2}(w, X_j)]$ tend to be 0 because of independent weights in PG and the weights having expectation 0. Hence, we focus on the strongly Gram part for the rest of the section. As in Equation (22), the number of terms in weakly Gram component grows quadratically in the width of the hidden layer. One might expect this results in a quadratic growth in convergence, given all the terms are i.i.d. random variables and strictly positive. In order to test this hypothesis, we exploit a teacher-student framework as outlined in Section 6, however the student models are 3-layer RELU-activated networks to ensure the number of terms in weakly Gram component is non-zero. We choose RELU activation function to ensure that the expected value of the activation to be greater than zero. Our experimental results shows that the convergence rate relationship with width is linear (see Figure 6) and not quadratic. This implies that weakly Gram components have expectation zero.

D Justify H stays close to initialization

Weights and H are indeed time-dependent and follow their respective stochastic process. However, theoretical results in the overparametrized regime suggest that every weight vector remains close to initialization (Lemma 5.3 (4) and Lemma 3.3 (13)). We also observe that the mean and variance stay very close to initialization, $\mathcal{N}(0, 0.1)$ (Figure 7). In Figure 8, we ran the Kolmogorov-Smirnov test to compare the weight distribution against the initialization distribution. As shown, the p-values are overall stable and far from significance levels to invalidate the null hypothesis (H_0: trained weights follow $\mathcal{N}(0, 0.1)$).

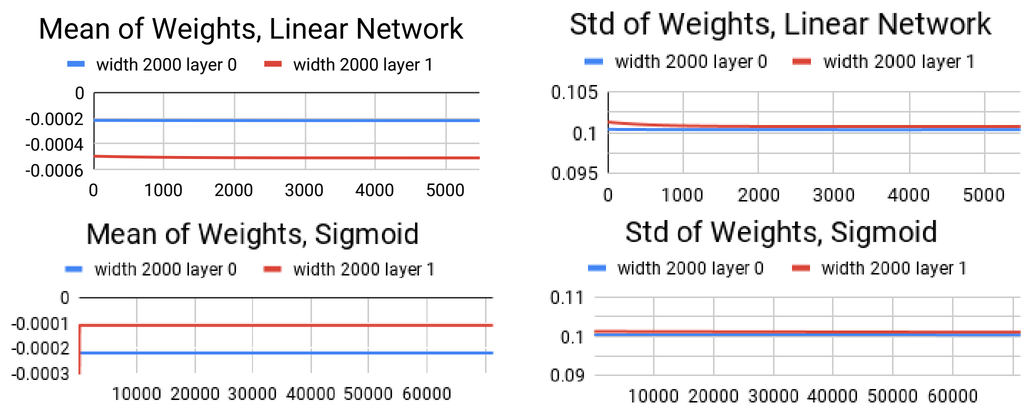


Figure 7: **Mean and std** across the training process. (X-axis is the number of steps)

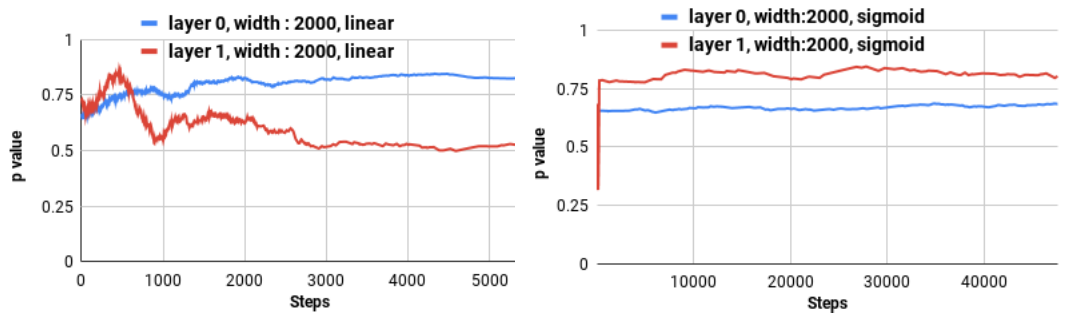


Figure 8: **KS test p-values** across the training process.

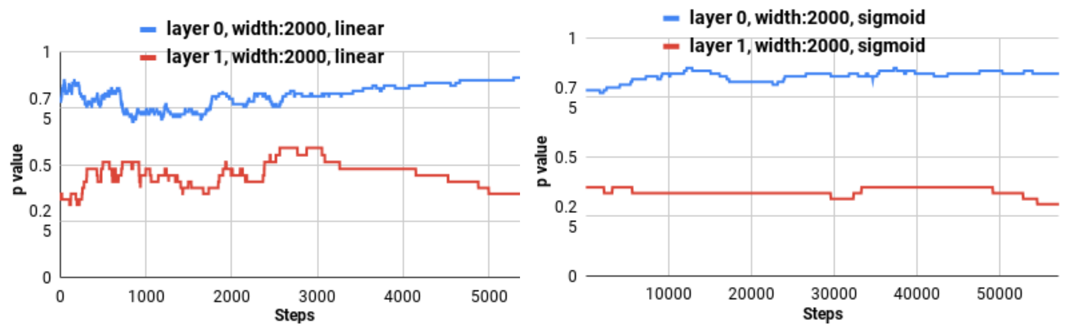


Figure 9: **Turning point test p-values**

E JUSTIFY IID ASSUMPTION FOR WEIGHTS

The Turning Point Test is commonly used to check iid-ness. Figure 9 shows the p-value through the training process. As shown, p-values do not exhibit any major regime shifts and the null hypothesis (the weights are iid samples) stays far from rejectable with any commonly used significance levels (0.1, 0.05 . . .). iid-ness can also be seen as a consequence of closeness to initialization in response 1. By staying close to initialization, statistical properties are preserved and the weights remain an iid sample of the original distribution.

F EIGENVALUE AND EIGENVECTOR OF K

In this section, we prove that K as defined in 5.2, we have the following:

Proposition F.1. *$K = (K_{ij}) = (g_i g_j)$ has one nonzero eigenvalue $(g_1^2 + ...g_N^2)$ with eigenvector $(g_1, ..., g_N)$*

Proof.

$$K = \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix} \times (g_1 \quad ... \quad g_n) \tag{36}$$

hence K has rank one. Therefore K one nonzero eigenvalue.

$$\begin{aligned} K \times \begin{pmatrix} g_1 \\ \vdots \\ g_d \end{pmatrix} &= \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix} \times (g_1 \quad ... \quad g_n) \times \begin{pmatrix} g_1 \\ \vdots \\ g_d \end{pmatrix} \\ &= \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix} \times \left((g_1 \quad ... \quad g_n) \times \begin{pmatrix} g_1 \\ \vdots \\ g_d \end{pmatrix} \right) = \begin{pmatrix} g_1 \\ \vdots \\ g_N \end{pmatrix} \cdot (g_1^2 + ...g_N^2) \end{aligned} \tag{37}$$

Hence the conclusion is true. □