Neural Path Features in Deep Learning

Chandrashekar Lakshminarayanan and Amit Vikram Singh,
Indian Institute of Technology, Palakkad

Abstract

Understanding optimisation and generalisation in deep neural networks (DNNs) with ReLU activations trained using first-order method such as gradient descent (GD) is an important problem in machine learning. To tackle this problem, we introduce a novel 'path-view': we decompose the computation in such DNNs into paths and gates. Here, we capture the state of the gates in a novel zeroth-order feature namely the *neural path feature* (NPF), whose dimension is equal to the total number of paths from the input to the output. Our work is related and complementary to prior work based on *neural tangent features and kernels* which are based on the first-order gradient information.

Using the 'path-view', we present a description of GD dynamics that incorporates NPF learning. We introduce a novel fixed NPF regime, in which, in the limit of infinite width, optimisation and generalisation properties can be tied down to an associated *neural path kernel*. In the fixed NPF regime, in the finite width case, we show that when optimising using GD i) increasing depth till a point helps in training and ii) increasing depth beyond a point hurts training. We show via experiments, that NPFs are learnt during training, and such learning improves generalisation performance.

1 Introduction

Understanding optimisation and generalisation in deep neural networks (DNNs) trained using first-order method such as gradient descent (GD) is an important problem in machine learning. Despite having a non-convex loss surface, GD achieves zero training error in over-parameterised DNNs where the number of parameters exceeds the size of the dataset. Interestingly, Zhang et al. [2016] demonstrated that practical DNNs have enough capacity to achieve zero training loss with even random labelling of standard datasets such as MNIST and CIFAR. However, when trained with true labels, such networks achieve zero training error and also exhibit good performance on test data.

In this paper, we consider fully connected DNNs with ReLU activations, with d layers and w hidden units per layer. In what follows ,we denote the dataset by $(x_s, y_s)_{s=1}^n \in \mathbb{R}^{d_{in}} \times \mathbb{R}$, and the parameters of the network by $\Theta \in \mathbb{R}^{d_{net}}$, the network output for an input $x \in \mathbb{R}^{d_{in}}$ by $\hat{y}_\Theta(x)$.

Some of the recent works (2**? ?** , 1) on this problem have made use of the *neural tangent features* (NTFs) and the *trajectory based analysis*, which we describe in brief. The NTF of an input $x \in \mathbb{R}^{d_{in}}$ is defined as $\psi_{x,\Theta} = (\partial_\theta \hat{y}_\Theta(x), \theta \in \Theta) \in \mathbb{R}^{d_{net}}$[1], i.e., the gradient of the network output with respect to the weights. By collecting the NTFs of all the inputs examples in the dataset, we can form the NTF matrix $\Psi = (\psi_{x_s,\Theta}, s \in [n]) \in \mathbb{R}^{d_{net} \times n}$[2]. The trajectory based analysis looks at the dynamics of the error $e_t = (\hat{y}_{\Theta_t} - y_s, s \in [n]) \in \mathbb{R}^n$. For a small enough step-size $\alpha_t > 0$ of the GD procedure, the error dynamics is given by:

$$e_{t+1} = e_t - \alpha_t K_{\Theta_t} e_t, \tag{1}$$

[1] Here $\partial_\theta(\cdot)$ stands for $\frac{\partial(\cdot)}{\partial\theta}$

[2] $[n] = \{1, \ldots, n\}$

Preprint. Under review.

where $K_{\Theta_t} \in \mathbb{R}^{n\times n}$ is the *neural tangent kernel* (NTK) matrix give by $K_\Theta = \Psi_\Theta^\top \Psi_\Theta$. Thus, the spectral properties, and in particular, $\rho_{\min}(K_{\Theta_t})$ the minimum eigenvalue of K_{Θ_t} dictates the rate of convergence. Under randomised initialisation, and in the limit of infinite width an interesting property emerges: the parameters of the DNN deviate very little during training, i.e. $\Theta_t \approx \Theta_0$. In particular, $K_{\Theta_t} \approx K_{\Theta_0}$, $K_{\Theta_0} \to K^{(d)}$, i.e., the NTK stays almost constant through training and the NTK matrix at initialisation K_{Θ_0} converges to a deterministic matrix $K^{(d)}$ (see Section 3 for exact expression of $K^{(d)}$. Thus in the 'large-width' regime, zero training error can be achieved if $\rho_{\min}(K^{(d)}) > 0$ which holds as long as the training data is not degenerate (2**?**). In the 'large-width' regime, **?** show that the fully trained DNN is equivalent to kernel regression with $K^{(d)}$. Hence, a trained DNN enjoys the generalisation ability of its corresponding $K^{(d)}$ matrix in the 'large-width' regime. **?** show that in the 'large-width' regime, the DNN is almost a linear learner with the random NTFs, and showed a generalisation bound in the form of $\tilde{\mathcal{O}}\left(d \cdot \sqrt{y^\top K^{(d)} y/n}\right)$[3], where $y = (y_s, s \in [n]) \in \mathbb{R}^n$ is the labelling function.

Research Gap I (Feature Learning): In the 'large-width' i.e., fixed NTF/NTK regime, the DNNs are linear learner using the random NTFs at random initialisation. This implies that there is little or no feature learning. Is this true?
Research Gap II (Finite vs Infinite): ? note that, while pure-kernel methods based on the limiting NTK (i.e., $K^{(d)}$) outperform other state-of-the-art kernel methods, the finite width DNNs (CNNs) still outperform their NTK (CNTK)[4] counterpart. Can we explain this gap?

2 Standard Regime: Capturing Feature Learning Via 'Path-View'

Notation: We consider fully connected deep networks with d layers, and w hidden units per layer, which accepts an input $x \in \mathbb{R}^{d_{in}}$, and produces an output $\hat{y}_\Theta \in \mathbb{R}$ where $\Theta \in \mathbb{R}^{d_{net}}$ ($d_{net} = w^2(d-2) + w(d_{in}+1)$). We denote by $\Theta(l,i,j)$ the weight connecting the i^{th} hidden unit of layer $l-1$ to the j^{th} hidden unit of layer l, and we use $G_{x,\Theta}(l,i)$ be the gating value $(0/1)$ of the i^{th} hidden unit in layer l for an input $x \in \mathbb{R}^{d_{in}}$.

2.1 Paths and Gates

We have a total of $P = d_{in} w^{(d-1)}$ paths. Let us say that an enumeration of the paths is given by $[P] = \{1, \dots, P\}$. Let $\mathcal{I}_l \colon [P] \to [w], l = 0, \dots, d-1$ provide the index of the hidden unit through which a path p passes in layer l (with the convention that $\mathcal{I}_d(p) = 1, \forall p \in [P]$). The activity of a path p for an input $x \in \mathbb{R}^{d_{in}}$ by $A_\Theta(x,p) \stackrel{def}{=} \Pi_{l=1}^{d-1} G_{x,\Theta}(l, \mathcal{I}_l(p))$.

In what follows, we consider two kinds of gates namely, i) hard gates, taking values in $\{0,1\}$, and ii) soft gates, taking values in $(0,1)$. For a pre-activation input $q \in \mathbb{R}$, the hard and soft gates are given by $\gamma_r(q) = \mathbb{1}_{\{\mathbf{q}>0\}}$ and $\gamma_{sr}(q) = \frac{1}{1+\exp(-\beta q)}$, where $\beta > 0$ is a positive constant. Using the hard and soft gates, we can specify the ReLU and 'soft-ReLU' activations as $\chi_r(q) = q \cdot \gamma_r(q)$ and $\chi_{sr}(q) = q \cdot \gamma_{sr}$. Here, in γ and χ, the subscripts 'r' and 'sr' stand for 'ReLU' and 'soft-ReLU' respectively.

2.2 Neural Path Feature and Values

The *neural path feature* (NPF) of an input $x \in \mathbb{R}^{d_{in}}$ is given by

$$\phi_{x,\Theta} \stackrel{def}{=} (x_s(\mathcal{I}_0(p)) A_\Theta(x_s, p), p \in [P]) \in \mathbb{R}^P, \tag{2}$$

where, for a path p, $\mathcal{I}_0(p)$ is the input node at which the path starts, and $A_\Theta(x_s,p)$ is its activity. The *neural path value* (NPV) if given by

$$v_\Theta \stackrel{def}{=} (\Pi_{l=1}^{d} \Theta(l, \mathcal{I}_{l-1}(p), \mathcal{I}_l(p)), p \in [P]) \in \mathbb{R}^P \tag{3}$$

[3] $a_t = \mathcal{O}(b_t)$ if $\limsup_{t\to\infty} |a_t/b_t| < \infty$, and $\tilde{\mathcal{O}}(\cdot)$ is used to hide logarithmic factors in $\mathcal{O}(\cdot)$.
[4] CNTK: Convolutional Neural Tangent Kernel, the NTK for CNNs

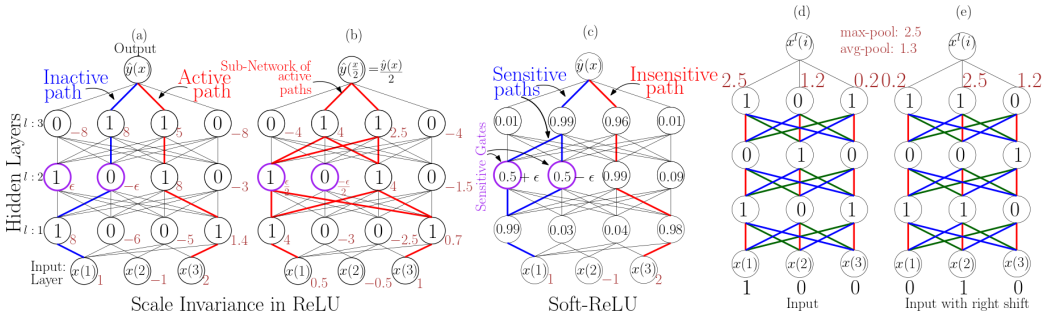


Figure 1: Cartoon illustration of usefulness of path-view and DGN framework.

The zeroth and first-order terms of a DNN can be written as:

$$\text{(Zeroth-Order)} \quad : \quad \hat{y}_\Theta(x) = \langle \phi_{x,\Theta}, v_\Theta \rangle = \sum_{p \in [P]} x(\mathcal{I}(p)) A_\Theta(x,p) v_\Theta(p) \tag{4}$$

$$\begin{aligned} \text{(First-Order)} \quad : \quad \partial \hat{y}_\Theta(x) &= \underbrace{\langle \phi_{x,\Theta}, \partial v_\Theta \rangle}_{\text{value gradient}} + \underbrace{\langle \partial \phi_{x,\Theta}, v_\Theta \rangle}_{\text{feature gradient}} \\ &= \sum_{p \in [P]} x(\mathcal{I}(p)) A_\Theta(x,p) \partial(v_\Theta(p)) + \sum_{p \in [P]} x(\mathcal{I}(p)) \partial(A_\Theta(x,p)) v_\Theta(p) \end{aligned} \tag{5}$$

Two learning problems and two gradients: (4) and (5) reveal that there are two learning problems namely i) learning the NPVs for a fixed NPF and ii) learning the NPFs. Corresponding to the two learning problems are the two gradients namely i) the value gradient $\langle \phi_{x,\Theta}, \partial v_\Theta \rangle$ which learns the NPVs, and ii) the feature gradient $\langle \partial \phi_{x,\Theta}, v_\Theta \rangle$ responsible for learning the NPFs.

Non-differentiability of ReLU: Note that $\langle \partial \phi_{x,\Theta}, v_\Theta \rangle = \sum_{p \in [P]} x(\mathcal{I}(p)) \partial(A_\Theta(x,p)) v_\Theta(p)$, and in the case of ReLU activations, the gating values are either 0 or 1, and hence $\partial(A_\Theta(x,p)) = 0$. However, the gating values and hence the path activities $A_{\Theta_t}(\cdot,\cdot)$ changes during training. This artefact arising due to non-differentiability are fixed by the *soft-ReLU* activation. This soft-ReLU 'trick' enables us to capture the terms related to feature learning in our analysis.

2.3 Information Flow via Sub-Networks

Active Sub-Networks and representational power: For each input example, a set of gates are *on* in each layer, and this gives rise to the sub-network of active paths for that input. This active sub-network can be said to hold the memory for a given input (see cartoon (b) in Figure 1). The NPFs capture this notion of active sub-network formally.

Zhang et al. [2016] showed that DNNs can fit even random labels, and random pixels of standard datasets such as MNIST. However, we note that, for DNNs with ReLU, with no bias parameters, a dataset with $n = 2$ points namely $(x, 1)$ and $(x/2, -1)$ for some $x \in \mathbb{R}^{d_{in}}$ cannot be memorised. The reason is that the gating values are the same for both x and $x/2$ (for that matter any positive scaling of x), and hence $\phi_{x/2,\Theta} = \phi_{x,\Theta}/2$, and thus it not possible to fit arbitrary values for $\hat{y}_t(x/2)$ and $\hat{y}_t(x)$, since $\hat{y}_t(x/2) = \hat{y}_t(x)/2$. Cartoons (a) and (b) in Figure 1 illustrate this scale invariance for inputs $x = (1, -1, 2) \in \mathbb{R}^3$ and $x/2 = (0.5, -0.5, 1) \in \mathbb{R}^3$: when compared to (a), pre-activation values and output of (b) are scaled down by a factor 2, and the gating values of (a) and (b) are identical.

Sensitive Sub-Network: In a DNN with soft-ReLU, $\partial A_\Theta(x,p) = \sum_{l=1}^{d} \partial G_{x,\Theta}(l) \Pi_{l' \neq l} G_{x,\Theta}(l')$ is significant for those paths p for which one of the $d-1$ gating values is close to 0.5 (say such a gate is in layer l, and for such a gate $\partial G_{x,\Theta}$ is significant) and rest of the $d-2$ gates are close to 1, so that $\Pi_{l' \neq l} G_{x,\Theta}(l')$ is significant. The set of paths for which $\partial A_\Theta(x,p)$ is significant, form the sensitive sub-network for that input. The sensitive paths are shown in cartoon (c) of Figure 1. In the case of, standard ReLU, sensitive paths are those which contain one of the gates with pre-activation close to 0, and rest of the $d-2$ gates are 1.

2.4 Gradient Dynamics with Feature Learning

Recall that the *neural tangent feature* (NTF) of an input $x \in \mathbb{R}^{d_{in}}$ is given by $\psi_{x,\Theta} = (\partial_\theta \hat{y}_\Theta(x), \theta \in \Theta) \in \mathbb{R}^{d_{net}}$. From (5) we have $\psi_{x,\Theta} = \psi^v_{x,\Theta} + \psi^\phi_{x,\Theta}$, where $\psi^v_{x,\Theta} = (\langle \phi_{x,\Theta}, \partial_\theta v_\Theta \rangle, \theta \in \Theta) \in \mathbb{R}^{d_{net}}$ and $\psi^\phi_{x,\Theta} = (\langle \partial_\theta \phi_{x,\Theta}, v_\Theta \rangle, \theta \in \Theta) \in \mathbb{R}^{d_{net}}$. Let $\Psi^v_\Theta = \left(\psi^v_{x_s,\Theta}, s \in [n]\right)$ and $\Psi^\phi_\Theta = \left(\psi^\phi_{x_s,\Theta}, s \in [n]\right)$ be $d_{net} \times n$ matrices, and let $K^v_\Theta = (\Psi^v_\Theta)^\top \Psi^v_\Theta$, $K^\phi_\Theta = (\Psi^\phi_\Theta)^\top \Psi^\phi_\Theta$, $K^{cross}_\Theta = (\Psi^v_\Theta)^\top \Psi^\phi_\Theta + (\Psi^\phi_\Theta)^\top \Psi^v_\Theta$ be $n \times n$ matrices. The GD dynamics with NPF learning can be given by:

Parameter Dynamics	$\dot{\Theta}_t$	$=$	$-\sum_{s=1}^n \psi_{x_s,t} e_t(s) = \sum_{s=1}^n (\psi^v_{x_s,t} + \psi^\phi_{x_s,t}) e_t(s)$
NPF Dynamics	$\dot{\phi}_{x_s,t}(p)$	$=$	$x(\mathcal{I}_0(p)) \sum_{\theta \in \Theta} \partial_\theta A_t(x_s, p) \dot{\theta}_t, \forall p \in [P], s \in [n]$
NPV Dynamics	$\dot{v}_t(p)$	$=$	$\sum_{\theta \in \Theta} \partial_\theta v_t(p) \dot{\theta}_t, \forall p \in [P]$
Kernel Decompostion	K_{Θ_t}	$=$	$K^v_{\Theta_t} + K^\phi_{\Theta_t} + K^{cross}_{\Theta_t}$
Error Dynamics	$\dot{e}_t$	$=$	$-K_{\Theta_t} e_t$

Table 1: Gradient Dynamics with NPF learning

• K^v_Θ is the kernel corresponding to learning NPVs. Let $\Phi_\Theta = (\phi_{x_s,\Theta}, s \in [n]) \in \mathbb{R}^{P \times d_{net}}$ be the NPF matrix, and define a $P \times d_{net}$ matrix with entries $\nabla_\Theta v_\Theta(p, \theta) = \partial_\theta v_\Theta$. Then, $K^v_\Theta = \Phi^\top_\Theta (\nabla_\Theta v_\Theta)(\nabla_\Theta v_\Theta)^\top \Phi^\top_\Theta$. Thus, as the NPF matrix Φ_{Θ_t} changes with time $K^v_{\Theta_t}$ changes with time.
• K^ϕ_Θ is the kernel corresponding to learning NPFs. Note that $K^\phi_\Theta = \sum_{\theta \in \Theta} \partial_\theta \Phi^\top_\Theta v v^\top \partial_\theta \Phi_\Theta$.

3 Large-Width Regime: Comparison with Prior Result

In the 'large-width' regime, as $w \to \infty$, in (**??**)we have $\dot{\Theta}_t, \dot{\phi}_t, \dot{v}_t \to 0$, and the error dynamics is given by

$$\dot{e}_t = -K^{(d)} e_t, \tag{6}$$

since, as $w \to \infty$, K_Θ in (**??**) converges to $K^{(d)}$ in (7) with $\chi = \chi_{sr}$.

$$\tilde{K}^{(1)}(s, s') = \Sigma^{(1)}(s, s') = \Sigma(s, s'), M^{(l)}_{ss'} = \begin{bmatrix} \Sigma^{(l)}(s, s) & \Sigma^{(l)}(s, s') \\ \Sigma^{(l)}(s', s) & \Sigma^{(l)}(s', s') \end{bmatrix} \in \mathbb{R}^2, \tag{7}$$

$$\Sigma^{(l+1)}(s, s') = 2 \cdot \mathbb{E}_{(q,q') \sim N(0, M^{(l)}_{ss'})} \left[\chi(q)\chi(q')\right], \hat{\Sigma}^{(l+1)}(s, s') = 2 \cdot \mathbb{E}_{(q,q') \sim N(0, M^{(l)}_{ss'})} \left[\partial\chi(q)\partial\chi(q')\right],$$

$$\tilde{K}^{(l+1)} = \tilde{K}^{(l)} \odot \hat{\Sigma}^{(l+1)} + \Sigma^{(l+1)},$$

where $s, s' \in [n]$ are two input examples in the dataset, Σ is the data Gram matrix, $\partial\chi$ stands for the derivative of the activation function with respect to the pre-activation input, $N(0, M)$ stands for the mean-zero Gaussian distribution with co-variance matrix M. The final limiting matrix is given by $K^{(d)} = \left(\tilde{K}^{(d)} + \Sigma^{(d)}\right)/2$.

In prior works, in the expression for the limit NTK $K^{(d)}$ in (7), χ was either χ_r (the standard ReLU) or χ_{sp} (known as the soft-plus), wherein, $\partial\chi_r$ is a Heaviside step function and $\partial\chi_{sp}$ is a logit function. Both these functions capture only the *on/off* behaviour of the gates, and hence $K^{(d)}$ captures only the flow of value gradient. On the contrary, in the case of soft-ReLU, $\partial\chi_{sr}$ contains both the *on/off* information via the $\partial\chi_{sp}$ term as well as the information on the dynamics of sensitive gates via the $\partial\gamma_{sr}$ term. Plugging the soft-ReLU instead of softplus/ReLU in (7) will indeed bring in the additional as below:

$$\hat{\Sigma}^{(l+1)}(s, s') = \hat{\Sigma}^{(l+1)}_v(s, s') + \hat{\Sigma}^{(l+1)}_\phi(s, s') + \hat{\Sigma}^{(l+1)}_{cross}(s, s'), \text{where}$$

$$\hat{\Sigma}^{(l+1)}_v(s, s') = 2 \cdot \mathbb{E}_{(q,q') \sim N(0, M^{(l)}_{ss'})} \left[\partial\chi_{sp}(q)\partial\chi_{sp}(q')\right],$$

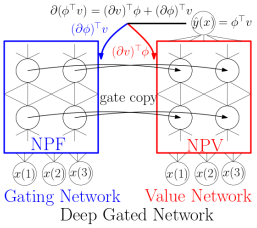
$$\hat{\Sigma}^{(l+1)}_\phi(s, s') = 2 \cdot \mathbb{E}_{(q,q') \sim N(0, M^{(l)}_{ss'})} \left[\partial\gamma_{sr}(q)\partial\gamma_{sr}(q')\right]$$

However, in the 'large-width' regime, $\dot{\theta}, \dot{\phi}, \dot{v}, \dot{K} \to 0$, and hence distinctions between $K^v_\Theta, K^\phi_\Theta$ does not make much difference. In this regime, only the fixed $K^{(d)}$ matters.

4 Fixed NPF regime

In the fixed NPF regime, we forcefully *turn-off* feature learning, i.e., we let the feature gradient $\psi^{\phi}_{x,\Theta} = 0$. In order to do this, we hold the gates and hence the NPFs in a separate gating network parameterised by $\Theta^g \in \mathbb{R}^{d_{net}}$, and the NPVs in a different network called value network parameterised by $\Theta \in \mathbb{R}^{d_{net}}$ (see Table 2). By letting the parameters of the gating network non-trainable, i.e., $\Theta^g_0 = \Theta^g_t, \forall t \geq 0$ we can force $\psi^{\phi}_{x,\Theta}$ to be 0. Thus, the gates are copied from the gating network onto the value network as shown on the right in Table 2.

Layer	Gating Network (NPF)	Value Network (NPV)
Input	$z_{x,\Theta^g_t}(0) = x$	$z_{x,\Theta_t}(0) = x$
Activation	$q_{x,\Theta^g_t}(l) = \Theta^g_t(l)^\top z_{x,\Theta^g_t}(l-1)$	$q_{x,\Theta_t}(l) = \Theta_t(l)^\top z_{x,\Theta_t}(l-1)$
Hidden	$z_{x,\Theta^g_t}(l) = \chi^G\left(q_{x,\Theta^g_t}(l)\right)$	$z_{x,\Theta_t}(l) = q_{x,\Theta_t}(l) \odot G_{x,\Theta^g_t}(l)$
Output	None	$\hat{y}_t(x) = \Theta_t(d)^\top z_{x,\Theta_t}(d-1)$
Gating Values: $G_{x,\Theta^g_t}(l) = \gamma_r\left(q_{x,\Theta^g_t}(l)\right)$ or $G_{x,\Theta^g_t}(l) = \gamma_{sr}\left(q_{x,\Theta^g_t}(l)\right)$		



Deep Gated Network

Table 2: Setup to learn with fixed NPFs.

A comparison of the GD dynamics across the three regimes namely i) the standard regime, wherein, width is finite, and the NPFs are learnt, ii) the 'large-width' regime, in which the NPFs, NTFs do not deviate much from initialisation and iii) the fixed NPF regime are given in Table 3. Note that if we let $w \to \infty$ in the fixed NPF regime, can have a regime which is both 'large-width' as well as fixed NPF (see middle region in Figure 2). Further, in the fixed NPF regime, $K_\Theta = K^v_\Theta$.

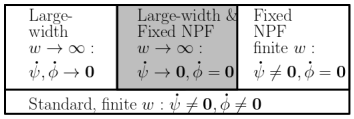


Figure 2: Regimes.

Finite w with NPF learning (standard)	Large-Width	Fixed NPF
$\dot{\Theta}_t = -\sum_{s=1}^n (\psi^v_{x_s,t} + \psi^{\phi}_{x_s,t}) e_t(s)$	$\dot{\Theta}_t \to 0$	$\dot{\Theta}_t = -\sum_{s=1}^n \psi^v_{x,t} e_t(s)$
$\dot{\phi}_{x_s,t}(p) = x(\mathcal{I}_0(p)) \sum_{\theta\in\Theta} \partial_\theta A_t(x_s,p)\dot{\theta}_t$	$\dot{\phi}_{x_s,t}(p) \to 0$	$\dot{\phi}_{x_s,t}(p) = 0$
$\dot{v}_t(p) = \sum_{\theta\in\Theta} \partial_\theta v_t(p)\dot{\theta}_t$	$\dot{v}_t(p) \to 0$	$\dot{v}_t(p) = \sum_{\theta\in\Theta} \partial_\theta v_t(p)\dot{\theta}_t$
$\dot{e}_t = -(K^v_t + K^{\phi}_t + K^{cross}_t) e_t$	$\dot{e}_t = -K^{(d)} e_t$	$\dot{e}_t = -K^v_t e_t$

Table 3: Dynamics in various regimes. Here $p \in [P], s \in [n]$.

Assumption 1 (Independent Initialisation). *(i) $\Theta_0 \in \mathbb{R}^{d_{net}}$ is statistically independent of NPFs, (ii) Θ_0 are sampled i.i.d from a distribution such that for any $\theta_0 \in \Theta_0$, we have $\mathbb{E}[\theta_0] = 0$, and $\mathbb{E}[\theta_0^2] = \sigma^2$, and $\mathbb{E}[\theta_0^4] = \sigma'^2$.*

Theorem 4.1 (Main Result). *Let $H_{\Theta^g_0} = \Phi^\top_{\Theta^g_0}\Phi_{\Theta^g_0}$ be the neural path kernel (NPK) matrix , then under Assumption 1, we have:*
(i) $\mathbb{E}[K_{\Theta_0}] = d\sigma^{2(d-1)} H_{\Theta^g_0}$.

(ii) In addition, if $4d/w^2 < 1$, then $Var[K_0] \leq O\left(d_{in}^2 \sigma^{4(d-1)} \max\{d^2 w^{2(d-2)+1}, d^3 w^{2(d-2)}\}\right)$.

Remark onAssumption 1 : This is known as the *independent initialisation* case, because, the NPVs v_{Θ_0} and NPFs $\phi_{x_s,\Theta^g_0}, s \in [n]$ are statistically independent at initialisation. Note that, in a standard DNN with ReLU activations, at initialisation, NPFs and NPV are derived from the same Θ_0, i.e., $\Theta^g_0 = \Theta_0$; we call this case as the *dependent initialisation* which does not satisfy Assumption 1. In the case of *dependent initialisation*, $K_{\Theta_0} \to K^{(d)}$, where $K^{(d)}$ can be calculated by letting $\chi = \chi_r$ in (7) (note that in the fixed NPF regime feature gradient is 0, and hence $\dot{\gamma} = 0$).

Remark on Theorem 4.1: This result says that in the fixed NPF regime, in the limit of infinite width, at random initialisation as per Assumption 1, the NTK K_{Θ_0} is equal to the NPK $H_{\Theta^g_0}$ times a scaling constant. From pervious results, **??**, if follows that as $w \to \infty$, the optimisation and generalisation properties of the fixed NPF learner can be tied down to $H_{\Theta^g_0}$. Theorem 4.1 can be applied in the following two interesting scenarios:

1. Learning with random fixed NPFs. Here, we choose Θ_0^g according to Assumption 1-(ii) and statistically independent of Θ_0. The learning problem in this case is $\hat{y}_{\Theta_t}(x) = \langle \phi_{x,\Theta_0^g}, v_{\Theta_t} \rangle$.
2. Learning with fixed NPFs from pre-trained networks. Here, we copy the weights of a pre-trained network into Θ_0^g, and then start with a random Θ_0 as Assumption 1, i.e., we retain the NPFs and then reset the NPVs to relearn them from scratch. Theorem 4.1 states that a network trained this way will generalise as well as its NPK.

Before we proceed to discuss the two scenarios, we will look at the structure of NPK matrix.

4.1 Hadamard Structure of NPK

Definition 4.1 (Correlation of Sub-Networks)**.** *Let* $\tau_\Theta(s, s', l) \stackrel{def}{=} \sum_{i=1}^{w} G_{x_s,\Theta}(l, i) G_{x_{s'},\Theta}(l, i)$ *be the number of activations that are* on *for both inputs* $s, s' \in [n]$ *in layer* l*. Let* $p \rightsquigarrow i$ *denote the fact that path* p *passes through input node* $i \in [d_{in}]$*, then define* $\Lambda_\Theta(s, s') \stackrel{def}{=} \sum_{p \rightsquigarrow i} A_\Theta(x_s, p) A_\Theta(x_{s'}, p)$*,* $\forall s, s' \in [n]$*, any* $i \in [d_{in}]$*.*

In the Definition 4.1 above, $\Lambda_\Theta \in \mathbb{R}^{n \times n}$ is the correlation matrix of the active sub-networks (consisting of the activations that are *on*) of different input pairs $s, s' \in [n]$. Note that the definition of Λ_Θ is not dependent on the choice of input node i, because, the terms inside the summation depend only on the path followed from the first layer onwards and excludes the input node.

Lemma 4.1. *It follows that:*

i) $\Lambda_\Theta(s, s') = \Pi_{l=1}^{d-1} \tau_\Theta(s, s', l)$.

ii) $H_\Theta = \Sigma \odot \Lambda_\Theta$*, where* $\odot$ *stands for the Hadamard product, and* $\Sigma \in \mathbb{R}^{n \times n}$ *is the input Gram matrix.*

We will now discuss about an interesting application of Theorem 4.1 to the problem of learning with fixed random NPFs.

4.2 Optimisation with fixed random NPFs at random initialisation: Role of Depth

Consider the case of independent initialisation, with $w \to \infty$. Here, $K_{\Theta_0} \to d\sigma^{(d-1)} H_{\Theta_0^g} = d(\Sigma) \odot (\sigma^{2(d-1)} \Lambda_{\Theta_0^g})$. Let us look at the diagonal and non-diagonal entries of $\sigma^{2(d-1)} \Lambda_0$, when Θ_0^g is initialised according to Assumption 1-(ii) for $\sigma = \sqrt{\frac{2}{w}}$.

• *Diagonal terms:* $\sqrt{\frac{2}{w}} \Lambda_{\Theta_0^g}(s, s) = 2^{(d-1)} \Pi_{l=1}^{(d-1)} \frac{\tau_{\Theta_0^g}(s, s', l)}{w}$. Note that as $w \to \infty$, $\frac{\tau_{\Theta_0^g}(s, s', l)}{w} \to \frac{1}{2}$, and hence $\sqrt{\frac{2}{w}} \Lambda_o(s, s) \to 1$.

• *Non-Diagonal terms:* As $w \to \infty$, $\sqrt{\frac{2}{w}} \Lambda_{\Theta_0^g}(s, s') = 2^{(d-1)} \Pi_{l=1}^{(d-1)} \frac{\tau_{\Theta_0^g}(s, s', l)}{w} \to \Pi_{l=1}^{(d-1)} \mu_l(s, s')$, where $\mu_l(s, s') \in (0, 1)$ is the fractional overlap between *on* activation in layer l for inputs $s, s' \in [n]$. Thus as depth increases the non-diagonal terms get suppressed by a factor $\Pi_l^{(d-1)} \mu_l(s, s')$.

When optimising with fixed NPFs, when the width is finite, increasing depth till a point helps in training due to the suppression of the non-diagonal terms of $H_{\Theta_0^g}$. Increasing depth beyond a point hurts, because, the variance term has a depth dependence, due to which, the entires of K_{Θ_0} deviate from their expected values.

5 Experiments: Fixed NPF Regime vs Standard Regime

The result of the numerical experiments are in Table 4. We used standard datasets namely MNIST, CIFAR-10, and CIFAR-100, with categorical cross entropy loss. The rows correspond to various architectures; Arch-1, Arch-2, Arch-3 and Arch-4 are fully connect DNNs with ReLU activations with $(w = 128, d = 6)$, $(w = 128, d = 10)$, $(w = 256, d = 6)$ and $(w = 256, d = 10)$ respectively. Arch-5 is a vanilla convolution neural network without pooling, residual connections, dropout or batch-normalisations, and is given by: input layer is $(32, 32, 3)$, followed by convolution layers with a stride of $(3, 3)$ and channels 64, 64, 128, 128 followed by a flattening to layer with 256 hidden

units, followed by a fully connected layer with 256 units, and finally a $10/100$ width soft-max layer to produce the final predictions. Rows 3 shows the result for standard ReLU network. Row $4, 5, 6$ and 7 shows the result for training with various fixed NPFs using the setup in Table 2. Row 4 shows the result for training with random fixed NPFs. Row 5 shows the result for fixed NPFs obtained a DNN with ReLU activations trained on datasets with true labels. Row 6 shows the result for fixed NPFs obtained a DNN with ReLU activations trained on dataset with random labels. Row 7 shows the result for fixed NPFs obtained a DNN with ReLU activations trained on dataset with random pixels. Note that in Row $5, 6, 7$, once the NPF is fixed, we train the NPVs, i.e., parameter Θ on a dataset with true labels.

Observations:
• Standard ReLU which learns the NPFs does better than fixed NPFs.

• Fixed NPFs obtained from DNNs trained on dataset with true labels perform as well as the standard ReLU. This means, once the NPFs are learned, we can copy the gates onto the gating network, turn off the feature learning, reset the weights corresponding to the NPV and relearn the NPVs from scratch to obtain performance. And from Theorem 4.1, we know that generalisation property is completely dictated by the NPF/NPK.

• Fixed NPFs obtained from DNNs trained on dataset with random labels/pixel perform worse than fixed random NPFs. Thus, bad labelling leads to learning bad NPFs.

NPF learning during training: We consider "Binary"-MNIST data set with two classes namely digits 4 and 7, with the labels taking values in $\{-1, +1\}$ and squared loss. We trained a standard DNN with ReLU activation ($w = 100$, $d = 5$). Let $\widehat{H}_t = \frac{1}{trace(H_t)} H_t$ be the normalised NPK matrix. For a subset size, $n' = 200$ (100 examples per class) we plot $\nu_t = y^\top (\widehat{H}_t)^{-1} y$, (where $y \in \{-1, 1\}^{200}$ is the labeling function), and observe that ν_t reduces as training proceeds (see Figure 3). Note that, $\nu_t = \sum_{i=1}^{n'} (u_{i,t}^\top y)^2 (\hat{\rho}_{i,t})^{-1}$, where $u_{i,t} \in \mathbb{R}^{n'}$ are the orthonormal eigenvectors of $\widehat{H}_t$ and $\hat{\rho}_{i,t}, i \in [n']$ are the corresponding eigenvalues. Since $\sum_{i=1}^{n'} \hat{\rho}_{i,t} = 1$, the only way ν_t reduces is when more and more energy gets concentrated on $\hat{\rho}_{i,t}$s for which $(u_{i,t}^\top y)^2$s are also high.

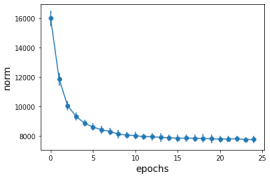


Figure 3: $\nu_t = y^\top (\widehat{H}_t)^{-1} y$.

References

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

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

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

	Dataset	ReLU	Twin	NPF(II)	NPF(DI)	NPF (trained	
						GL	RL
Arch-1	MNIST	98.14 ± 0.07	98.22 ± 0.05	96.02 ± 0.13	96.09 ± 0.12	97.82 ± 0.02	95.21 ± 0.1
Arch-1-SGD	MNIST	97.85 ± 0.09	97.86 ± 0.11	95.85 ± 0.10	95.85 ± 0.17	97.10 ± 0.09	$\pm$
Arch-2	MNIST	98.11 ± 0.08		95.69 ± 0.12	95.54 ± 0.13	98.11 ± 0.05	93.40 ± 0.1
Arch-2-SGD	MNIST	97.80 ± 0.08		95.76 ± 0.07	95.81 ± 0.09	97.22 ± 0.08	$\pm$
Arch-3	MNIST	98.42 ± 0.05		96.81 ± 0.12	96.90 ± 0.14	98.33 ± 0.06	94.92 ± 0.1
Arch-3SGD	MNIST	98.05 ± 0.06		96.23 ± 0.13	96.63 ± 0.14	97.54 ± 0.08	$\pm$
Arch-4	MNIST	98.41 ± 0.06		96.52 ± 0.11	96.71 ± 0.22	98.45 ± 0.06	
Arch-4-SGD	MNIST	97.98 ± 0.05		95.81 ± 0.32	$96.25 \pm 0.0.05$	97.70 ± 0.13	
Arch-5	CIFAR-10	67.02 ± 0.43		58.92 ± 0.62	58.83 ± 0.27	63.06 ± 0.73	

Table 4: Shows the training and generalisation performance of various NPFs.