Duality simplifies deep neural networks with rectified linear units

Anonymous Author(s)
Affiliation
Address
email

Abstract

Our aim is an improved and simplified understanding of the inner workings of deep neural network (DNNs) with rectified linear units (ReLUs). In particular, we focus on the gating property of ReLU, i.e., it either blocks (multiplies by 0) or allows (multiplies by 1) its pre-activation input. Due to the gating property, for each input there is an *active/on* sub-network comprising of those gates which are *on* and the weights between those gates. Recently, Lakshminarayanan and Singh [2020] developed a *dual view*, where the DNN with ReLUs is broken into paths so as to separate the gates from the weights. Using the dual view they showed that, most information is in the gates, and that the gates are analytically characterised by a *neural path kernel* (NPK) which is a Hadamard product of the input Gram matrix and a correlation matrix measuring the size of the overlapping active sub-networks.

In this paper, we present three theoretical and three experimental results. The main theoretical result is that each layer has a *base kernel* measuring the *correlation of gates*, and the NPK is a *Hadamard* product of these *base kernels* and the input Gram matrix. To our knowledge, this is the simplest kernel in literature that analytically characterises the gates. First experimental result (to justify the theoretical result) is that, operations destroying the layer-by-layer structure such as permuting the layers, arbitrarily tiling and rotation of the gates and providing a constant input do not degrade performance, because, in all these operations, the correlation of gates is not lost. Secondly, we show experimentally that, (open question related to) the degradation in test accuracy due to upstream training with random labels can be 'root caused' to the gates – this adds strength to the dual view and the importance of gates. Thirdly, we modify standard architectures (VGG19 and a ResNet) to yield two deep gated networks in which feature extraction is free of activations and is separate from the gates and the weights – these achieve greater than 90% test accuracy on CIFAR-10. The other two theoretical results extend the dual view to cover the cases of convolutions with pooling and skip connections.

1 Introduction

Understanding the inner workings of deep neural networks (DNNs) is an important problem in machine learning. Recent works [12, 1, 4] have connected the training and generalisation of DNNs to kernel methods. An important kernel associated with a DNN is its *neural tangent kernel* (NTK), which, for a pair of input examples $x, x' \in \mathbb{R}^{d_{\text{in}}}$, and network weights $\Theta \in \mathbb{R}^{d_{\text{net}}}$, is given by:

$$\text{NTK}(x, x') \quad = \quad \langle \nabla_\Theta \hat{y}(x), \nabla_\Theta \hat{y}(x') \rangle, \quad \text{where}$$

$\hat{y}_\Theta(\cdot) \in \mathbb{R}$ is the DNN output. It was shown that, as the width of the DNN goes to infinity, the NTK matrix converges to a limiting deterministic matrix NTK_∞, and training an infinitely wide DNN is equivalent to a kernel method with NTK_∞. While these recent results allows us to look

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

at DNNs from the lens of kernels, there are some important issues: (i) **feature learning:** NTK_∞ being a deterministic matrix does not capture feature learning whereas the success of DNNs is due to feature learning, (ii) **finite vs infinite width:** finite width convolutional neural network outperforms its corresponding NTK_∞ and (iii) **non-interprability:** the NTK is the inner product of gradients and has no physical interpretation. As a result, NTK theory does not fully explain the success of DNNs.

Lakshminarayanan and Singh [2020] aimed at understanding DNNs with rectified linear units (ReLUs). Such DNNs have a special property in that each ReLU is also a gate which either blocks (multiplies by 0) or allows its pre-activation input (multiplies by 1). Using the *dual view*, they characterised the role of gating analytically and empirically, showing that most useful information is learnt in the gates. The dual view also successfully addresses the issues of **feature learning, finite vs infinite width and non-interpretability** present in the NTK theory as described below.

• **Dual View:** The standard primal way of expressing information flow in a DNN is layer by layer. In the dual view, gating property of ReLU is exploited to break the DNN into paths. A path comprises of gates and weights, and a path is 'active' or 'on' only if all the gates in the path are 'on'.

• **Measure of information in the gates:** The gates are treated as masks and are decoupled from the weights by storing the gates and weights in two separate networks. The information in the gates is measured by fixing the gates, training only the weights and looking at the test performance.

• **Gates = Features:** When the gates from a trained DNN are used as masks, and the weights are trained from scratch, there is no significant loss in test performance, i.e., **features are in the gates**. Also, the performance of NTK_∞ lies between the performance of untrained and trained gates, i.e., **learning in the gates explains the difference between finite vs infinite** width DNNs with ReLUs.

• **Interpretable Kernel:** Each input has a corresponding *active* sub-network (comprising of the gates that are 'on' and weights between such gates) which produces the output. When the gates are decoupled from the weights, the NTK simplifies into a constant times a *neural path kernel* (NPK):

$$\text{NTK}(x, x') \quad \propto \quad \text{NPK}(x, x') \quad = \quad \langle x, x' \rangle \cdot \mathbf{overlap}(x, x'), \tag{1}$$

where $\mathbf{overlap}(x, x')$ is a correlation matrix that measures the overlap of active sub-networks in terms of the total number of paths that are 'active' for both inputs x and x'.

1.1 Our Contribution

In this paper, we present six results (first three are theoretical, followed by experimental) that strengthen the dual view and the importance of gates. The first experimental result verifies the theoretical insights. The second experimental result settles an open problem using the dual view showcasing its usefulness. The final result builds on the insights obtained from the first five results.

1. **Product Kernel:** Expression in (1) is in terms of the active sub-network overlap. We further simplify this down to the gates to show that the NPK has the following product structure:

$$\text{NTK}(x, x') \propto \langle x, x' \rangle \cdot \Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x') \rangle}{w},$$

where $G_l(x) \in \{0, 1\}^w$ is the binary feature encoding the gates of layer l. The main message is that the *base kernels* $\frac{\langle G_l(x), G_l(x') \rangle}{w}$ measuring the ***correlation of gates*** captures the role of gates.

2. **Convolutional layers with pooling** makes the NPK *rotationally invariant*.

3. **Skip connections** endow a *sum of product of base kernels* structure to the NPK.

4. **Robustness of Gates:** Operations such as permutation of layers, arbitrary tiling cum rotation of the gates and providing a constant $\mathbf{1}$ input, which we design to destroy the traditional layer-by-layer flow, do not degrade performance since the 'correlation of gates' is not lost in these operations.

5.**Gates and Random Labels:** Upstream training with random labels followed by downstream training with true labels degrades test performance [21]. We show that this degradation is due to the fact that random labels affect the gates, and gates do resist such degradation to a large extent. Also, if the gates are fixed, upstream training by random labels does not degrade test performance.

6. **No Hidden Features:** We modify standard architectures (VGG19 and a ResNet) to yield two deep gated networks in which feature extractor is free of activations and is separate from the gates and the weights – these achieve greater than 90% test accuracy on CIFAR-10. Since the feature extractor uses only standard 'image processing' operations, the features are interpretable and no longer hidden.

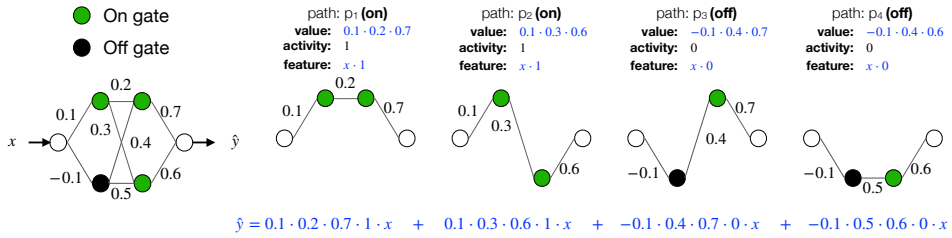


Figure 1: Illustration of Definition 2.1 and Proposition 2.1 in a toy network with 2 layers, 2 gates per layer and 4 paths. Paths p_1 and p_2 are 'on' and paths p_3 and p_4 are 'off'. The value, activity and feature of the individual paths are shown. $\hat{y}$ is the summation of the individual path contributions.

2 Preliminaries on the Dual View for DNN with ReLUs

In this section, we will present a brief overview of the dual view for a fully connected (FC) DNN with ReLUs as presented in [15]. The main points are: (i) the neural path feature (encoding the gates and the input) and the neural path value (encoding the weights) in Definition 2.1 and the expression (2) in Proposition 2.1, where the output of the DNN is equal to the summation of individual path contributions (illustrated in Figure 1), (ii) the correlation matrix of active sub-network overlaps in Definition 2.2 and its relation to the *neural path kernel* (this is the Gram matrix of the neural path features) in Lemma 2.1 and (iii) the deep gated network which decouples the gates and the weights.

Notation. In what follows, $[n]$ denotes the set $\{1, \ldots, n\}$.

2.1 Gates, Weights and Sub-Networks: Neural Path – Features, Value and Kernel

We consider a FC-DNN with 'd' layers and w hidden units in each layer.

Definition 2.1. *A path starts from an input node, passes through a weight and a hidden unit in each layer and ends at the output node. We define the following quantities for a path p:*

Activity	:	$A_\Theta(x,p)$ is the product of the '$d-1$' gates in the path.
Value	:	$v_\Theta(p)$ is the product of the 'd' weights in the path.
Feature	:	$\phi_{x,\Theta}(p)$ is the product of the signal at the input node of the path and $A_\Theta(x,p)$.

There are $P^{\text{fc}} = d_{\text{in}} w^{(d-1)}$ paths and a path is active only if all the gates in the path are active.

Proposition 2.1. *Assuming that the paths can be enumerated as $[P^{fc}]$, one can collect the features and values of all the paths in the so called* neural path feature *(NPF) given by $\phi_{x,\Theta} = (\phi_{x,\Theta}(p)), \in [P^{fc}]$ and the* neural path value *(NPV) given by $v_\Theta = (v_\Theta(p)), \in [P^{fc}]$. The output of the DNN is then the inner product of the NPF and NPV, i.e.,*

$$\hat{y}_\Theta(x) = \langle \phi_{x,\Theta}, v_\Theta \rangle = \sum_{p \in [P]} \phi_{x,\Theta}(p) v_\Theta(p) \tag{2}$$

Definition 2.2 (Overlap of active sub-networks)**.** *The total number of 'active' paths for both x and x' that pass through input node i is defined to be:*

$$\textbf{\textit{overlap}}_\Theta(i,x,x') = \Lambda_\Theta(i,x,x') \triangleq \left| \{p \in [P] \colon \mathcal{I}_0^f(p) = i, A_\Theta(x,p) = A_\Theta(x',p) = 1\} \right|$$

Lemma 2.1 (Neural Path Kernel (NPK))**.** *Let $D \in \mathbb{R}^{d_{in}}$ be a vector of non-negative entries and for $u, u' \in \mathbb{R}^{d_{in}}$, let $\langle u, u' \rangle_D = \sum_{i=1}^{d_{in}} D(i) u(i) u'(i)$. Let $H_\Theta(x,x') \triangleq \langle \phi_{x,\Theta}, \phi_{x',\Theta} \rangle$ be the neural path kernel (NPK). Then*

$$NPK_\Theta(x,x') = H_\Theta(x,x') = \langle x, x' \rangle_{\Lambda_\Theta(\cdot,x,x')}$$

Remark. In the case of fully connected networks, $\textbf{overlap}_\Theta(i,x,x')$ is equal for all $i \in [d_{\text{in}}]$, and hence $\text{NPK}_\Theta(x,x') = \langle x, x' \rangle \cdot \textbf{overlap}_\Theta(x,x')$.

2.2 Deep Gated Network : Decoupling Gates (NPF) and Weights (NPV)

Since $\hat{y}_\Theta(x) = \langle \phi_{x,\Theta}, v_\Theta \rangle$, during training, as Θ is learnt, both the NPFs and NPV are also learnt. To understand their roles better, $\phi_{x,\Theta}$ and v_Θ have to be separated. This is achieved by the deep

gated network (DGN) setup (see Figure 2), which has two networks of *identical architecture* namely the *feature network* ($\Theta^{\mathrm{f}} \in \mathbb{R}^{d^{\mathrm{f}}_{\mathrm{net}}}$) which holds the NPFs (i.e., the gating information) and the *value network* ($\Theta^{\mathrm{v}} \in \mathbb{R}^{d^{\mathrm{v}}_{\mathrm{net}}}$) which holds the NPV. The combined parameterisation is denoted by $\Theta^{\mathrm{DGN}} = (\Theta^{\mathrm{f}}, \Theta^{\mathrm{v}}) \in \mathbb{R}^{d^{\mathrm{f}}_{\mathrm{net}}+d^{\mathrm{v}}_{\mathrm{net}}}$. The feature network is a DNN with ReLUs and the value network is a DNN with *Gated Linear Units (GaLUs)* (terminology used in [8]) whose output is the product of its pre-activation input $q^{\mathrm{v}}(x)$and the external gating signal $G^{\mathrm{f}}(x)$ (see Figure 2). In Figure 2, the main output of the DGN is $\hat{y}_{\mathrm{DGN}}(x)$, while the other output $\hat{y}_{\mathrm{f}}(x)$ is used to *pre-train* the gates (see Section 4).

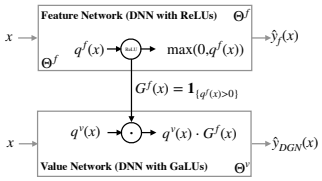


Figure 2: Shows a DGN and interaction between two corresponding units, one in the feature and other in value network.

Proposition 2.2 ([15]). *Let* $K^{v}_{\Theta^{DGN}}(x,x') = \langle \nabla_{\Theta^v} \hat{y}_{DGN}(x), \nabla_{\Theta^v} \hat{y}_{DGN}(x') \rangle$, *and* $K^{f}_{\Theta^{DGN}}(x,x') = \langle \nabla_{\Theta^f} \hat{y}_{DGN}(x), \nabla_{\Theta^f} \hat{y}_{DGN}(x') \rangle$ *and* $K_{\Theta^{DGN}}(x,x')$ *be the NTK of the DGN. Then*

$$K_{\Theta^{DGN}}(x,x') = K^{v}_{\Theta^{DGN}}(x,x') + K^{f}_{\Theta^{DGN}}(x,x')$$

Remarks. *$\nabla_{\Theta^f} \hat{y}_{DGN}(x)$ flows via the feature network and $\nabla_{\Theta^v} \hat{y}_{DGN}(x)$ via the value network. Correspondingly there are two NTKs, i.e., K^f and K^v. If the gates are fixed then, $\nabla_{\Theta^f} \hat{y}_{DGN}(x) = \mathbf{0}$ and $K^{f}_{\Theta^{DGN}}(x,x') = 0$.*

3 Theoretical Result

Our goal is to characterise 'what is learnt in the gates of a DNN with ReLUs'. To this end, we use the DGN setup where the gates and weights are separate. Specifically, the DNN with ReLUs whose gates we want to examine is the feature network in the DGN. We then measure the information in the gates by keeping the gates fixed and training the value network and measuring the test performance of the value network. It was shown in [15] that if the gates are fixed (i.e., with reference to Proposition 2.2, $K^{\mathrm{f}} = 0$), for fully connected DGN, the NTK simplifies as $\mathrm{NTK}(x,x') \propto \mathrm{NPK}(x,x') = \langle x, x' \rangle \cdot \mathbf{overlap}(x,x')$. In this section, we state three results, where in Theorem 3.1 we 're-write' Theorem 5.1 in [15] to explicitise the role of gates, width and depth, to yield a simple product of kernels expression (unnoticed in prior works). Theorem 3.2 covers the case of convolutions with pooling and Theorem 3.3 covers the case residual networks with skip connections.

Note. The gates of the DGN can also be tuned, in which case $K^{\mathrm{f}} \neq \mathbf{0}$, and we reserve the study of K^{f} for future. The results are for infinite width networks and at randomised initialisation. We use these insights as an indicator of what we can expect when we experiment with finite width networks.

We begin with an assumption that states that the value network weights have to be statistically independent of the feature network weights at initialisation.

Assumption 3.1. $\Theta^{v}_{0} \overset{i.i.d}{\sim}$ *Bernoulli*$(\frac{1}{2})$ *over* $\{-\sigma, +\sigma\}$ *and statistically independent of* Θ^{f}_{0}.

We point out that this statistical decoupling of weights and gates in Assumption 3.1 is unrealisable in a DNN with ReLU, however, this assumption can be trivially realised in a DGN.

Theorem 3.1 (Product of Kernels Theorem)**.** *Under Assumption 3.1 ($\sigma = \frac{c_{scale}}{\sqrt{w}}$) for FC-DGN :*

$$K^{v}_{\Theta^{DGN}_{0}}(x,x') \to d \cdot c^{d-1}_{scale} \cdot \left(\langle x, x' \rangle \cdot \Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x') \rangle}{w} \right), \quad as\ w \to \infty$$

Key Insight. Here $\frac{\langle G_l(x), G_l(x') \rangle}{w}$ are the *base kernels* measuring the ***correlation of the gates***. We show experimentally that the correlation of gates is essentially 'what is learnt in a DNN with ReLUs'.

Feature Network. The role of this network is to process the input layer-by-layer and produce the w-dimensional gating features $G_l(\cdot)$. Each layer comprises of 'w' ReLUs, and a given ReLU (i.e., gate) is 'on' if the input to that layer lies on the positive half-space of hyperplane of given by the incoming weights of that ReLU. Thus the gates of a given layer are based on the angle between the input to that layer and the various hyperplanes given by the weights of that layer. Prior experiments in [15] and the experiments in this paper show that the feature network, i.e., the gates hold most information, which in turn means that weights of the feature network are key.

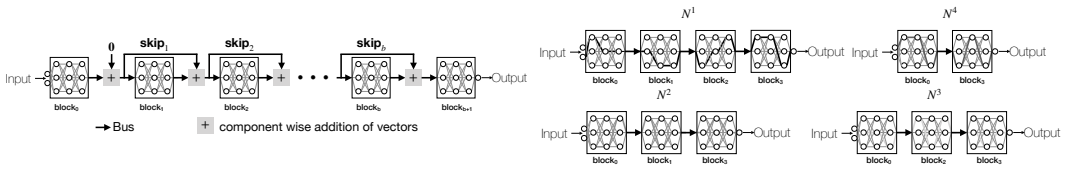


Figure 3: On the left is the ResNet with b skip connections and $(b+2)$ blocks. On the right are the sub-FC networks N^1 obtained by skipping no blocks, N^2 and N^4 obtained by skipping block 1 and 2 respectively, and N^3 obtained by skipping both blocks 1 and 2.

Value Network. The value network implements the product of kernels by laying out the gates as masks depth-wise, and connecting them in the structure of a DNN. Note that depth-wise layout plays an important role here: for instance, if we were to concatenate the gating features as $\varphi(x) = (G_l(x), l = 1, \ldots, d-1) \in \{0,1\}^{(d-1)w}$, it would have only resulted in the kernel $\langle \varphi(x), \varphi(x') \rangle = \sum_{l=1}^{d-1} \langle G_l(x), G_l(x') \rangle$, i.e., a *sum (not product)* of kernels. Prior experiments in [15] and the experiments in this paper show that the value network can be reset and re-trained without loss of performance, which in turn means that weights of the value network are not that important.

Note. The above insights from Theorem 3.1 carry over to the case of DNN with ReLUs by thinking that the roles of value and feature network are performed by a single network.

Convolutions with pooling. Let the circular rotation of vector $x \in \mathbb{R}^{d_{\text{in}}}$ by 'r' co-ordinates be defined as $rot(x,r)(i) = x(i+r)$, if $i + r \leq d_{\text{in}}$ and $rot(x,r)(i) = x(i+r-d_{\text{in}})$ if $i + r > d_{\text{in}}$. Using circular convolutions with pooling results in a rotationally invariant kernel Theorem 3.2. The architecture and the notations for the network with convolutions is presented in the Appendix.

Theorem 3.2 (Rotationally Invariant Kernel Theorem)**.** *Under Assumption 3.1, for a suitable β_{cv}:*

$$K^{v}_{\Theta_0^{DGN}} \rightarrow \frac{\beta_{cv}}{{d_{in}}^2} \cdot \sum_{r=0}^{d_{in}-1} \langle x, rot(x',r) \rangle_{\Lambda(\cdot,x,rot(x',r))}, \text{ as } w \rightarrow \infty \text{ (for global-average-pooling)},$$

$$K^{v}_{\Theta_0^{DGN}} \rightarrow \beta_{cv} \cdot \sum_{r=0}^{d_{in}-1} \langle x, rot(x',r) \rangle_{\Lambda(\cdot,x,rot(x',r))}, \text{ as } w \rightarrow \infty \text{ (for global-max-pooling)}$$

Remark. $\sum_{r=0}^{d_{\text{in}}-1} \langle x, rot(x',r) \rangle_{\Lambda(\cdot,x,rot(x',r))} = \sum_{r=0}^{d_{\text{in}}-1} \Big(\sum_{i=1}^{d_{\text{in}}} x(i) rot(x',r)(i) \Lambda(i,x,x') \Big)$, where the inner '$\Sigma$' is the inner product between x and $rot(x',r)$ weighted by Λ and the outer 'Σ' covers all possible rotations, which results in the rotational invariance property.

Residual Networks with Skip connections. We consider a ResNet with '$(b+2)$' blocks and 'b' skip connections between the blocks (left of Figure 3). Each block is a fully connected (FC) DNN of depth 'd_{blk}' and width 'w'. There are combinatorially many sub-FC-DNNs within this ResNet (see Definition 3.1 and right of Figure 3).

Definition 3.1. *[Sub FC-DNNs] Let $2^{[b]}$ denote the power set of $[b]$ and let $\mathcal{J} \in 2^{[b]}$ denote any subset of $[b]$. Define the '$\mathcal{J}^{th}$' sub-FC-DNN of the ResNet to be the fully connected network obtained by (i) including block$_j, \forall j \in \mathcal{J}$ and (ii) ignoring block$_j, \forall j \notin \mathcal{J}$ (see Figure 3).*

Theorem 3.3 (Sum of Product of Kernels Theorem)**.** *Let $H^{\mathcal{J}}_{\Theta_0^f}$ be the NPK of the $\mathcal{J}^{th}$ sub-FC-DNN, and $\beta^{\mathcal{J}}_{fc}$ be the associated constant. Under Assumption 3.1, we have:*

$$K^{v}_{\Theta_0^{DGN}} \rightarrow \sum_{\mathcal{J} \in 2^{[b]}} \beta^{\mathcal{J}}_{fc} H^{\mathcal{J}}_{\Theta_0^f}, \text{ as } w \rightarrow \infty$$

Note. The sub-FC-DNNs we refer to in Theorem 3.3 belong to (or are parts of) the feature network from which the gates are obtained.

4 Numerical Experiments

In what follows, Section 4.1 is themed around the key insight obtained from Theorem 3.1, i.e., the correlation of the gates lies at the heart of 'what is learnt in a DNN with ReLUs'. In particular, we

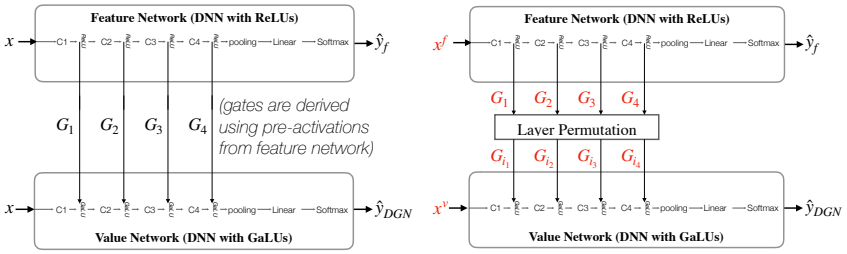


Figure 4: The prior setup in [15] is shown in the left and the setup in this paper is shown in the right. Here $C1, \ldots, C4$ are convolutional layers with 128 output filters, kernel size 3×3 and stride 1×1.

Dataset	Fixed Random II	Fixed Random DI	Decoupled Learning	Fixed Learnt	ReLU (Feature Network)
MNIST[16]	$94.1_{\pm 0.3}$	$94.1_{\pm 0.3}$	$98.1_{\pm 0.1}$	$98.6_{\pm 0.1}$	$98.5_{\pm 0.1}$
CIFAR-10[14]	$67.5_{\pm 0.7}$	$67.6_{\pm 0.7}$	$77.6_{\pm 0.6}$	$79.4_{\pm 0.3}$	$80.4_{\pm 0.3}$

Table 1: Shows the % test accuracy of various gates. The main result here is that the numbers in columns 1 to 4 are averaged over 48 models (1 run per model, best performance in each run) and performance is robust to layer permutations and $x^{\text{v}} = \mathbf{1}$ input. For ReLU the average is over 5 independent runs. Optimiser: Adam (3e-4).

show that operations (such as permuting the layers, tiling cum rotation of the gates, and giving a constant 'all-ones' input to the value network) that destroy the layer by layer computational structure do not degrade test performance. These operations lead to combinatorially many models and in all of them the test performance remains the same. In Section 4.2 we throw light on the open question in [21] on why test performance degrades due to upstream training with random labels.

4.1 Robustness of Gates: Destroying layer-by-layer structure does not affect performance

In this experiment we show that information in the gates is invariant even if we destroy the layer-by-layer structure. We consider the three kinds of gates (or gating regimes) described below.

1. **Fixed Learnt:** We *pre-train* the feature network (which is a DNN with ReLUs), then *freeze* the weights of the feature network, and then train the value network. This way we can measure information in the gates of a trained DNN.
2. **Fixed Random:** We initialise the feature network at random, and then *freeze* its weights and then train the value network. This way we can measure information in the gates of a DNN at initialisation. Here, the feature and value network can be either initialised with the same weights i.e., dependent initialisation (DI) or statistically independent weights i.e., independent initialisation (II).
3. **Decoupled Learning:** We initialise the weights of the value and feature network statistically independent and then train both of them. For the gradient to flow through the feature network we use *soft-gating*, i.e., $G(q) = \frac{1}{1+\exp(-\beta \cdot q)}$, with $\beta = 10$.

We make use of the deep gated network (DGN) setup shown in Figure 4 (right) which is an improvisation of the DGN in prior work [15] shown in Figure 4 (left). In the current setup, $G_{i_1}, \ldots, G_{i_4}$ **is a permutation of the** $G_1, \ldots, G_4$, which gives 24 models. Note that these **permutations destroy the layer by layer structure**. In the prior setup both value and feature networks have the same input $x \in \mathbb{R}^{d_{\text{in}}}$. In the current setup, we have two separate inputs, $x^{\text{f}} \in \mathbb{R}^{d_{\text{in}}}$ for the feature network and $x^{\text{v}} \in \mathbb{R}^{d_{\text{in}}}$ for the value network. We set $x^{\text{f}} = x$ always, however, for x^{v} there are **two modes** namely (i) **standard**: we set $x^{\text{v}} = x$,(ii) **'all-ones'**: we set $x^{\text{v}} = \mathbf{1} \in \mathbb{R}^{d_{\text{in}}}$ (a tensor of all 1's). While the setup on the left is only one model, the setup on the right has $\mathbf{48 = 24 \times 2}$ **different models**, where $24 = \texttt{factorial}(4)$ is due to the gate permutations and 2 is due the settings of x^{v}.

Note on prior results. The following two observations were made in prior work [15]: (i) by using the gates and training the NPV from scratch we can recover the performance (see ReLU vs Fixed Learnt in Table 1), and (ii) the performance of Convolutional NTK (77.43%)[1] is between that of finite width network with random gates (67.5%) and learnt gates give (79.4%), i.e., learning in the gates explains the difference between finite width DNNs and infinite width NTK.

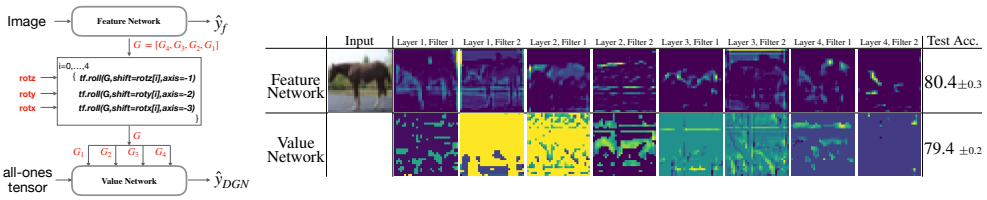


Figure 5: On the left is the setup for arbitrary rotation of gates. On the right, we have the images related to feature and value networks on top and bottom rows respectively. Each row has the input image followed by the output of first 2 filters in each of the 4 layers. In the bottom row, the input column 'appears' blank because it is the image of the $\mathbf{1}$ input to the value network. Despite the arbitrary rotation and $\mathbf{1}$ input, the value network is within 1% of feature network's test accuracy (on CIFAR-10).

For 'all-ones' input to value network, in Theorem 3.1, only $\langle x, x' \rangle = d_{\text{in}}$, and $\Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x') \rangle}{w}$ is still unchanged. Similar explanations holds for Theorems 3.2 and 3.3.

Robustness of Gates. Entries in columns 1 to 4 in Table 1 are averaged over 48 different models which included permuting the layers and providing $\mathbf{1}$ as input. Our main result is that the information in the gates is robust over these 48 models – these justify the insights from Theorem 3.1 that as long as correlation in the gates is not destroyed the performance does not degrade. We also went one step further with respect to destroying the structure by considering arbitrary rotations. Here, we first tile the gates of the various filters (as a tensor) in the reverse order starting from layer 4 to layer 1 and then rotate arbitrarily in each of the three dimension for five times as shown in left of Figure 5 (i.e, two image dimensions and one filter dimension). For each rotation in the filter dimension we choose a random integer belonging to $[0, 512)$ and for each of the rotation in the image dimensions we choose a uniform random integer belonging to $[0, 32)$. We did 10 independent runs of the 'fixed learnt' gates and 'decoupled learning' of gates and observed the test performance to be $79.4_{\pm 0.2}$ and $79.0_{\pm 0.5}$ respectively (which are similar to the numbers shown in Table 1).

Primal vs Dual View of features. The standard (primal) view is that the input is processed layer-by-layer and the hidden features are in the layer outputs. Our experiments challenge the primal view, because, we do not provide the image as input but only $\mathbf{1}$ ('all-ones') as input to the value network, and we have destroyed the layer-by-layer structure of the gates before applying them as masks in the value network as well (also, the 'fixed learnt' gates do not change during training). As a result, there is a huge 'visual' difference between the hidden layer outputs of the feature network and that of the value network (see right of Figure 5). One could still insist that DNNs are so powerful that they are recovering the features layer-by-layer. Or alternatively, one can appeal to the dual view to conclude that since neither 'all-ones' input nor the arbitrary rotation of the gates affect the correlation of gates (Theorem 3.1), the performance does not degrade. While the gates are generated layer-by-layer in the feature network, when applied as masks, the function of the gates is to select and train the sub-networks corresponding to each example (viewpoint trivially following from Proposition 2.1) .

Gate learning need not be layer-by-layer. In 'decoupled learning' (Table 1), while the gates are generated layer-by-layer in the feature network, after arbitrary rotations they end in a different location in the value network during training, i.e., the gates can be arbitrarily placed while learning.

4.2 Upstream training with random labels and downstream with true labels affects the gates

Q1 (open question [21]). When trained with random labels upstream followed by true labels downstream, the test performance of a DNN with ReLUs degrades, Why?

Setup to answer Q1. We hypothesise that the answer to the above question lies in the gates. To test our hypothesis, we train in two phases (i) Phase I: upstream training with label noise levels $\gamma = 0, 25\%, 50\%, 75\%$ and (ii) Phase II: downstream training with true labels. For $\gamma = 0$ there is no Phase II because Phase I is already with true labels. We then measure the information stored in the gates at the end of each of the two phases. To this end, we consider the DGN setup in Figure 4 (left), and train the feature network (which is a DNN with ReLUs) with $\hat{y}_{\text{f}}$ as output node. In Phase I, we train the feature network for different values of label noise $\gamma = 0, 25\%, 50\%, 75\%$, which gives us models $M1(\gamma)$ (see Figure 6). To measure the information in the gates learnt at the end of Phase I, we keep these gates fixed and train the value network with true labels – this gives us models $M2(\gamma)$

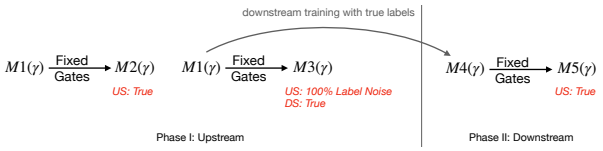


Figure 6: Shows various models trained in Experiment 2. Here, US and DS stand for upstream and down stream respectively. True for training with true labels. Here models $M1$ and $M4$ are feature networks (DNNs with ReLUs), and $M2$, $M3$ and $M5$ are value networks which use the fixed gates from the feature networks. Models are parameterised by (γ) which is the label noise level.

	Phase I: Upstream				Phase II: Downstream	
	ReLU		Fixed Gates		ReLU	Fixed Gates
γ	Best	End	True	US&DS	True	True
0	**81.2**$\pm$ 0.3	80.0$\pm$ 0.4	**80.3**$\pm$ 0.2	**79.3**$\pm$ 0.4	-	-
25	76.1$\pm$ 0.6	63.1$\pm$ 0.7	74.7$\pm$ 0.5	72.3$\pm$ 0.2	76.9$\pm$ 0.1	76.9$\pm$ 0.4
50	70.9$\pm$ 0.8	41.5$\pm$ 1.1	69.9$\pm$ 0.3	66.8$\pm$ 0.1	73.4$\pm$ 0.3	73.6$\pm$ 0.4
75	56.9$\pm$ 0.4	23.4$\pm$ 0.4	63.9$\pm$ 0.4	60.0$\pm$ 0.3	68.1$\pm$ 0.4	67.7$\pm$ 0.6
Models	M1(γ)		M2(γ)	M3(γ)	M4(γ)	M5(γ)

Table 2: Shows the % test accuracy (on test data with true labels) of the models in Experiment 2 on CIFAR-10. The numbers are averaged over 3 runs (best accuracy is taken in each run except for column 'End'). For column 'End', in each run the average test accuracy over the last 10 epochs are considered. Optimiser: Adam(3e-4).

(see Figure 6). In Phase II, we start with models $M1(\gamma)$ (i.e., obtained at end of Phase I), and perform downstream training with true labels to obtain models $M4(\gamma)$. To measure the information in the gates learnt at the end of Phase II, we keep these gates fixed and train the value network with true labels – this gives us models $M5(\gamma)$ (see Figure 6).

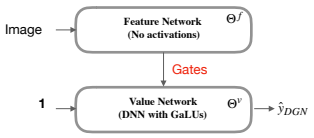
Q2 (new question). When trained with random labels upstream followed by true labels downstream, *if the gates are fixed throughout and only the weights are trained*, does the test performance degrade?

Setup to answer Q2. Here, we pick up the models trained at the end of Phase I, use them as feature networks and fix the gates. We then train the value network, first upstream with with 100% random labels, and then downstream with 100% true labels–this gives us models $M3(\gamma)$.

The results of the experiments with random labels shown in Table 2 (plots are in the Appendix).

Answer to Q2. *When the gates are fixed test performance is robust to upstream training with random labels.* In Table 2, look at the performance of $M3(\gamma = 0)$ and compare it with performance of $M4(\gamma), \gamma = 25, 50, 75$. When the gates are fixed, i.e., in $M3(0)$ (note: this is a value network), the upstream training with even 100% label noise does not hurt the test performance so much $(81.2 - 79.3 < 2\%)$ in comparison to the cases when the gates are allowed to change, i.e., $M4(\gamma)$ (note: these are feature networks), the test performance degrades (from 81.2) to 76.9 (4.3% for $\gamma = 25$), 73.6 (7.6% for $\gamma = 50$) and 68.1 (13.1% for $\gamma = 75$) for even less than 100% label noise.

Answer to Q1. Recall from Proposition 2.1 that in a DNN with ReLUs (which is the feature network), $\hat{y}_{\Theta^{\mathrm{f}}}(x) = \langle \phi_{x,\Theta^{\mathrm{f}}}, v_{\Theta^{\mathrm{f}}} \rangle$, and when training, both $\phi_{x,\Theta^{\mathrm{f}}}$ and $v_{\Theta^{\mathrm{f}}}$ are learnt. We now examine how random labels affect $\phi_{x,\Theta^{\mathrm{f}}}$ and $v_{\Theta^{\mathrm{f}}}$. During upstream training, the test performance initially improves, reaches a 'Best' and then degrades (see 'Best' and 'End' in Table 2). Now, if we use the NPFs $\phi_{x,\Theta^{\mathrm{f}}}$ (i.e., gates) at the end of Phase I and train the NPV ($v_{\Theta^{\mathrm{v}}}$) separately in the value network to obtain $M2(\gamma)$ we see that $M2(\gamma)$ is better than the 'Best' of $M1(\gamma)$. This implies degradation post the 'Best' value mostly affects only the NPV ($v_{\Theta^{\mathrm{f}}}$) and not the NPFs. Yet, the NPFs $\phi_{x,\Theta^{\mathrm{f}}}$ at end of Phase I, i.e., $M2(\gamma), \gamma = 25, 50, 75$ are worser than $M2(0)$. Now, even when we perform downstream training with true labels in Phase II, the recovery is not full (i.e., $M4(\gamma)$ are not close to $M1(0) = 81.2$). After both Phases I and II, both $\phi_{x,\Theta^{\mathrm{f}}}$ and $v_{\Theta^{\mathrm{f}}}$ have gone through upstream as well as downstream training, and the lack of recovery could be due to either $\phi_{x,\Theta^{\mathrm{f}}}$ or $v_{\Theta^{\mathrm{f}}}$. In order to separate this we take only the NPFs $\phi_{x,\Theta^{\mathrm{f}}}$ (i.e., gates at end of Phase II) and retrain the NPV ($v_{\Theta^{\mathrm{v}}}$) afresh with true labels in the value network to obtain $M5(\gamma)$. Now from the fact that $M5(\gamma) \approx M4(\gamma)$, we know that the NPFs $\phi_{x,\Theta^{\mathrm{f}}}$ (i.e., the gates) are the real reason for the degradation. We can conclude that (i) *the degradation in test performance is due to the gates (i.e., NPFs), and (ii) gates (i.e., NPFs) do resist such degradation (i.e., after the 'Best' the NPFs do not degrade).*



	VGG19	ResNet
Standard DNN	$93.4_{\pm 0.1}$	$93.0_{\pm 0.1}$
DGN-No-Act	$91.7_{\pm 0.1}$	$90.9_{\pm 0.1}$

Figure 7: On the left is the DGN-No-Act. On right is % test accuracy on CIFAR-10, averaged over 5 runs (best in each run). Optimiser: SGD with momentum of 0.9 (the learning rate and schedule are in the Appendix).

5 A Simple And Interpretable Deep Gated Network: No Hidden Features

We used the DGN setup to answer the question 'what is learnt in a DNN with ReLUs'. We now ask: 'What happens if we remove activations (i.e., replace it with identity activations) from the feature network of a DGN?'– let us call this the DGN-No-Act. If the activations are removed from the feature network, then feature generation will no longer be hidden, i.e., it will only be in terms of transformations such as matrix multiplication, pooling and scale/mean correction (in the case of batch norm), all of which have concrete 'image processing' interpretations. The gates in DGN-No-Act are derived (as usual) from the pre-activations of the identity activations in the feature network. We use soft-gating $G(q) = \frac{1}{1+\exp(-\beta \cdot q)}$, with $\beta = 10$. The value network in the DGN-No-Act has the same functionality, i.e., it learns the NPV. We constructed two such DGN-No-Acts based on standard architectures (VGG19 [24] and a ResNet[23]), wherein, the standard architecture is used in feature network (without ReLU) and in the value network with ReLU replaced by GaLU. Both these DGN-No-Acts gave more than 90% test accuracy on CIFAR-10 (see Figure 7).

6 Related Works

Kernels. Several works have examined theoretically as well as empirically two important kernels associated with a DNN namely its NTK based on the correlation of the gradients and the conjugate kernel based on the correlation of the outputs [7, 10, 20, 6, 26, 5, 12, 1, 22, 17, 18]. In contrast, the NPK is based on the correlation of the gates, and instead of designing a pure-kernel method with NPK, we use the NPK as an aid to experimentally examine gates of a finite width DNN with ReLUs. The dynamics of NTK is captured analytically using an infinite hierarchy of differential equations in [11], and empirically by a set of diverse measures in [9]. In contrast, in the dual view (our paper and in [15]), the dynamics of the gates is captured by having the feature network to be either untrained, partially/fully trained. It was shown in [19], that prediction using CNTK with GAP is equivalent to prediction using CNTK without GAP but with full translation data augmentation with wrap-around at the boundary. This is related to Theorem 3.2. It was shown in [25] that residual networks behave like ensemble of shallow networks. This is related Theorem 3.3.

Random Labels. In [21], both positive and negative effects on downstream training performance due to upstream training with random labels was studied. The question of why the test performance degrades due to upstream training with random labels was left open, which we addressed in our paper.

ReLU. A spline theory based on max-affine linearity was proposed in [3] to show that a DNN with ReLUs performs hierarchical, greedy template matching, and this framework was extended in [2] to cover many non-linear activations. In contrast, the dual view exploits the gating property to simplify the NTK into the NPK. Gated linearity was studied in [8] for single layered networks, along with a non-gradient algorithm to tune the gates. In contrast, we look at networks of any depth, and the gates are tuned via standard optimisers. Another novelty in our work in comparison to the above works is the DGN-No-Act where feature generation is activation free, explicit and not hidden.

7 Conclusion

We showed that the correlation of the gates captures 'what is learnt by a DNN with ReLUs', and that operations that destroy layer-by-layer structure do not degrade performance as long as the correlation of the gates is not lost. We also showed that upstream training with random labels affects the gates which is the reason for degradation of test performance even after down stream training with true labels. We also proposed a deep gated network in which feature extraction was free from activations, therefore, explicit and not hidden, and achieved greater than 90% test accuracy on CIFAR-10.

8 Broader Impact

This paper argues that the correlation of the *on/off* states of the rectified linear units holds the key in understanding of inner workings of deep neural networks with rectified linear units. This paper also proposes a deep gated network in which the feature generation is carried out without using any activation functions, and is separate from the gates and the weights, thereby clearly demarcating the role of the different parts. We believe that this work will be a useful addition towards understanding deep neural networks and making them more interpretable and explainable.

References

[1] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019.

[2] Randall Balestriero and Richard G Baraniuk. From hard to soft: Understanding deep network nonlinearities via vector quantization and statistical inference. *arXiv preprint arXiv:1810.09274*, 2018.

[3] Randall Balestriero et al. A spline theory of deep learning. In *International Conference on Machine Learning*, pages 374–383, 2018.

[4] Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10835–10845, 2019.

[5] Shuxiao Chen, Hangfeng He, and Weijie J Su. Label-aware neural tangent kernel: Toward better generalization and local elasticity. *arXiv preprint arXiv:2010.11775*, 2020.

[6] Zixiang Chen, Yuan Cao, Quanquan Gu, and Tong Zhang. A generalized neural tangent kernel analysis for two-layer neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[7] Zhou Fan and Zhichao Wang. Spectra of the conjugate kernel and neural tangent kernel for linear-width neural networks. *arXiv preprint arXiv:2005.11879*, 2020.

[8] Jonathan Fiat, Eran Malach, and Shai Shalev-Shwartz. Decoupling gating from linearity. *CoRR*, abs/1906.05032, 2019. URL `http://arxiv.org/abs/1906.05032`.

[9] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *arXiv preprint arXiv:2010.15110*, 2020.

[10] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Ronen Basri. On the similarity between the laplace and neural tangent kernels. *arXiv preprint arXiv:2007.01580*, 2020.

[11] Jiaoyang Huang and Horng-Tzer Yau. Dynamics of deep neural networks and neural tangent hierarchy. In *International Conference on Machine Learning*, pages 4542–4551. PMLR, 2020.

[12] 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.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[15] Chandrashekar Lakshminarayanan and Amit Vikram Singh. Neural path features and neural path kernel: Understanding the role of gates in deep learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[16] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[17] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[18] Jaehoon Lee, Samuel S Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *arXiv preprint arXiv:2007.15801*, 2020.

[19] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

[20] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33, 2020.

[21] Hartmut Maennel, Ibrahim Alabdulmohsin, Ilya Tolstikhin, Robert JN Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. What do neural networks learn when trained with random labels? *arXiv preprint arXiv:2006.10455*, 2020.

[22] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148*, 2018.

[23] David C. Page. Davidnet. URL `https://github.com/davidcpage/cifar10-fast`.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 2016.

[26] Lechao Xiao, Jeffrey Pennington, and Samuel Schoenholz. Disentangling trainability and generalization in deep neural networks. In *International Conference on Machine Learning*, pages 10462–10472. PMLR, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes] The fact that we are studying DNNs with only ReLUs has been made explicit in the title as well as through the entire paper. Also, our specific focus are the gates under the decoupling assumption which is discussed in Section 3.

(c) Did you discuss any potential negative societal impacts of your work? [N/A]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes]

(b) Did you include complete proofs of all theoretical results? [Yes] In the Appendix.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the code as part of supplementary material. The dataset are standard.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Tables 1 and 2 and Fig. 7

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Tables 1 and 2 and Fig. 7

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We describe the experiments in a bit more detail in the Appendix, where we also mention the computational resources used.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We are using standard datasets namely CIFAR-10 and MNIST and we have included the citations for the same.

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Numerical Experiments

We now list the details related to the numerical experiments which have been left out in the main body of the paper.

• **Computational Resource.** All the numerical experiments were run in 'Google Colaboratory' ("https://colab.research.google.com") and the models were in *Tensorflow*.

• For experiments in Section 4 we used Adam [13] with learning rate of 3×10^{-4}, and batch size of 32.

• In Section 4.1, for the results Table 1 for MNIST we used the DGN architectures in Figure 4 with fully connected layers instead of convolutional layers.

• The training and test accuracy plots for the random label experiments in Section 4.2 are shown in Figures 8 and 9. In particular, the fact that while training with random labels upstream the performance reaches a peak ('Best') and then it degrades ('End') can be seen in $M1(\gamma)$ in the top row of Figure 8.

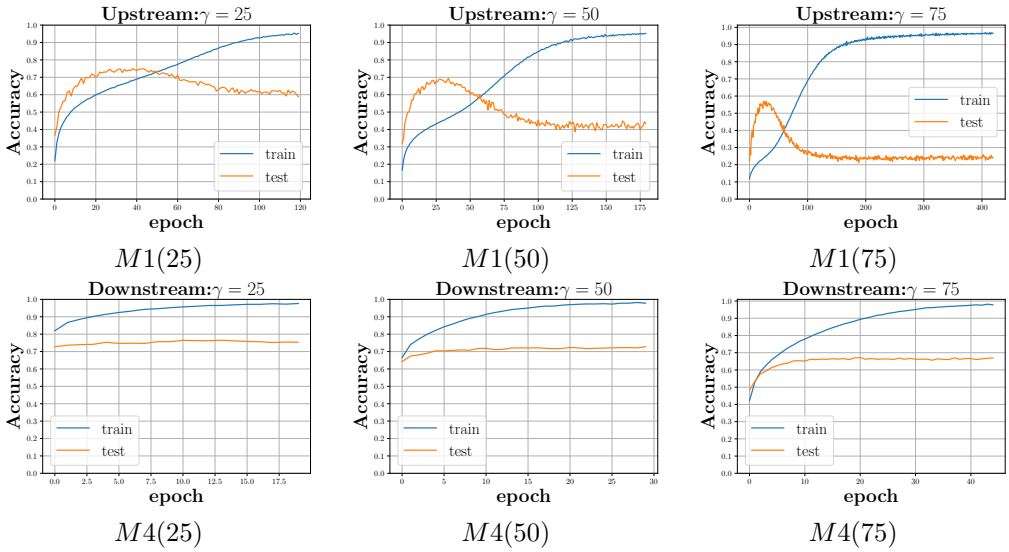


Figure 8: Shows the upstream training with random labels $M1(\gamma)$ followed downstream training with true labels $M4(\gamma)$. The first epochs of bottom plots is same as the epoch follows the last epoch in the top plots.

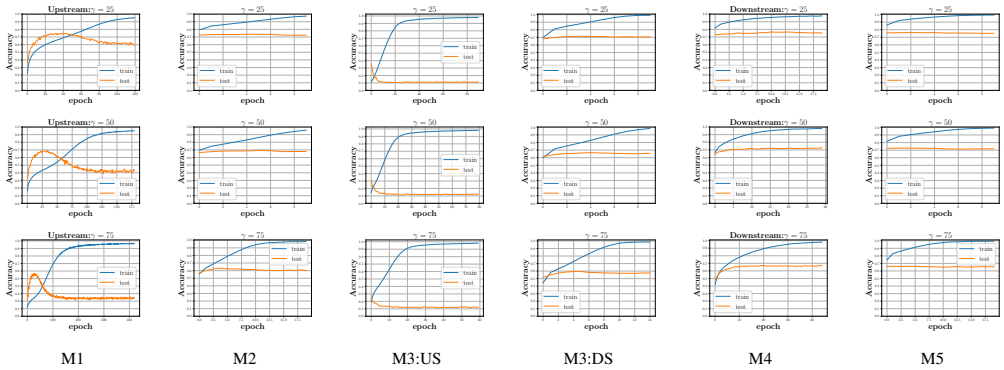


Figure 9: Shows the training and test accuracy plots of a single run for models $M1, M2, M3, M4$ and $M5$.

• In Section 5, the code for VGG19 architecture is from the repository "https://github.com/gahaalt/resnets-in-tensorflow2", and the ResNet architecture called `DavidNet` is from the repository "https://github.com/davidcpage/cifar10-fast". The `DGN-No-Acts` based on these architectures were derived in a manner described in Section 5.

• For VGG19 (and the corresponding `DGN-No-Act`), we used *SGD* optimiser with momentum 0.9 and the following learning rate schedule (as suggested in "https://github.com/gahaalt/resnets-in-tensorflow2") : for iterations $[0, 400)$ learning rate was 0.01, for iterations $[400, 32000)$ the learning rate was 0.1, for iterations $[32000, 48000)$ the learning rate was 0.01, for iterations $[48000, 64000)$ the learning rate was 0.001. The batch size was 128. The models were trained till 32 epochs.

• For `DavidNet` (and the corresponding `DGN-No-Act`), we used *SGD* optimiser with momentum 0.9, learning rate 0.4, batch size of 512, weight decay of 5×10^{-4} (as suggested in "https://github.com/davidcpage/cifar10-fast").

B Fully Connected

We first present the formal definition for the neural path features and neural path values for the fully connected case.

The layer-by-layer way of expressing the computation in a DNN of width 'w' and depth 'd' is given below.

Input Layer	:	$z_{x,\Theta}(\cdot,0)$	$=$	x
Pre-Activation	:	$q_{x,\Theta}(i_{\text{out}},l)$	$=$	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}},i_{\text{out}},l)\cdot z_{x,\Theta}(i_{\text{in}},l-1)$
Gating	:	$G_{x,\Theta}(i_{\text{out}},l)$	$=$	$\mathbf{1}_{\{q_{x,\Theta}(i_{\text{out}},l)>0\}}$
Hidden Layer Output	:	$z_{x,\Theta}(i_{\text{out}},l)$	$=$	$q_{x,\Theta}(i_{\text{out}},l)\cdot G_{x,\Theta}(i_{\text{out}},l)$
Final Output	:	$\hat{y}_\Theta(x)$	$=$	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}},i_{\text{out}},d)\cdot z_{x,\Theta}(i_{\text{in}},d-1)$

Table 3: Information flow in a FC-DNN with ReLU. Here, 'q's are pre-activation inputs, 'z's are output of the hidden layers, 'G's are the gating values. $l\in[d-1]$ is the index of the layer, i_{out} and i_{in} are indices of nodes in the current and previous layer respectively.

Notation B.1. *Index maps identify the nodes through which a path p passes. The ranges of index maps $\mathcal{I}_l^f$, $\mathcal{I}_l, l\in[d-1]$ are $[d_{in}]$ and $[w]$ respectively. $\mathcal{I}_d(p)=1,\forall p\in[P^{fc}]$.*

Definition B.1. *Let $x\in\mathbb{R}^{d_{in}}$ be the input to the DNN. For this input,*

(i) $A_\Theta(x,p)\triangleq\Pi_{l=1}^{d-1}G_{x,\Theta}(\mathcal{I}_l(p),l)$ is the activity of a path.

(ii) $\phi_{x,\Theta}\triangleq\left(x(\mathcal{I}_0^f(p))A_\Theta(x,p),p\in[P^{fc}]\right)\in\mathbb{R}^{P^{fc}}$ is the neural path feature (NPF).

(iii) $v_\Theta\triangleq\left(\Pi_{l=1}^{d}\Theta(\mathcal{I}_{l-1}(p),\mathcal{I}_l(p),l),p\in[P^{fc}]\right)\in\mathbb{R}^{P^{fc}}$ is the neural path value (NPV).

We now restate Theorem 3.1 in the notation introduced in Table 3.

Theorem B.1 (Product of Kernels Theorem)**.** *Under Assumption 3.1 ($\sigma=\frac{c_{scale}}{\sqrt{w}}$) for FC-DGN :*

$$K^v_{\Theta_0^{DGN}}(x,x')\overset{(a)}{\to}d\cdot\sigma^{2(d-1)}\cdot H_{\Theta_0'}(x,x'),\quad \text{as } w\to\infty$$
$$\overset{(b)}{=}d\cdot c_{scale}^{2(d-1)}\cdot\left(\langle x,x'\rangle\cdot\Pi_{l=1}^{d-1}\frac{\langle G_{x,\Theta}(\cdot,l),G_{x',\Theta}(\cdot,l)\rangle}{w}\right)$$

Proof. Here (a) is same as Theorem 5.1 in [15]. From from Lemma 2.1 we know that $H_\Theta(x,x')=\langle x,x'\rangle_{\Lambda_\Theta(\cdot,x,x')}$, and that in the case of fully connected networks $\Lambda_\Theta(i,x,x')$ does not change with i. (b) follows from the fact the total number of paths simultaneous 'active' is equal to the product of the total number of gates simultaneously active, i.e., $\Lambda_\Theta(x,x')=\langle G_{x,\Theta}(\cdot,l),G_{x',\Theta}(\cdot,l)\rangle$. □

C Convolution with pooling

Architecture: We consider (for sake of brevity) a 1-dimensional convolutional neural network with circular convolutions, with d_{cv} convolutional layers ($l=1,\dots,d_{\text{cv}}$), followed by a *global-average/max-pooling* layer ($l=d_{\text{cv}}+1$) and d_{fc} ($l=d_{\text{cv}}+2,\dots,d_{\text{cv}}+d_{\text{fc}}+1$) FC layers. The convolutional window size is $w_{\text{cv}}<d_{\text{in}}$, the number of filters per convolutional layer as well as the width of the FC is w.

Indexing: Here $i_{\text{in}}/i_{\text{out}}$ are the indices (taking values in $[w]$) of the input/output filters. i_{cv} denotes the indices of the convolutional window taking values in $[w_{\text{cv}}]$. i_{fout} denotes the indices (taking values in $[d_{\text{in}}]$, the dimension of input features) of individual nodes in a given output filter. The weights of layers $l\in[d_{\text{cv}}]$ are denoted by $\Theta(i_{\text{cv}},i_{\text{in}},i_{\text{out}},l)$ and for layers $l\in[d_{\text{fc}}]+d_{\text{cv}}$ are denoted by $\Theta(i_{\text{in}},i_{\text{out}},l)$. The pre-activations, gating and hidden unit outputs are denoted by $q_{x,\Theta}(i_{\text{fout}},i_{\text{out}},l)$, $G_{x,\Theta}(i_{\text{fout}},i_{\text{out}},l)$, and $z_{x,\Theta}(i_{\text{fout}},i_{\text{out}},l)$ for layers $l=1,\dots,d_{\text{cv}}$.

Definition C.1 (Circular Convolution)**.** *For $x\in\mathbb{R}^{d_{in}}$, $i\in[d_{in}]$ and $r\in\{0,\dots,d_{in}-1\}$, define :*

(i) $i\oplus r=i+r$, for $i+r\le d_{in}$ and $i\oplus r=i+r-d_{in}$, for $i+r>d_{in}$.

IL	:	$z_{x,\Theta}(\cdot, 1, 0)$	$=$	x
Convolutional Layers, $l \in [d_{\text{cv}}]$				
PA	:	$q_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$	$=$	$\sum_{i_{\text{cv}}, i_{\text{in}}} \Theta(i_{\text{cv}}, i_{\text{in}}, i_{\text{out}}, l) \cdot z_{x,\Theta}(i_{\text{fout}} \oplus (i_{\text{cv}} - 1), i_{\text{in}}, l-1)$
GV	:	$G_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$	$=$	$\mathbf{1}_{\{q_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l) > 0\}}$
HUO	:	$z_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$	$=$	$q_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l) \cdot G_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$
GAP Layers, $l = d_{\text{cv}} + 1$				
HUO	:	$z_{x,\Theta}(i_{\text{out}}, d_{\text{cv}} + 1)$	$=$	$\sum_{i_{\text{fout}}} z_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, d_{\text{cv}}) \cdot G^{\text{pool}}_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, d_{\text{cv}} + 1)$
Fully Connected Layers, $l \in [d_{\text{fc}}] + (d_{\text{cv}} + 1)$				
PA	:	$q_{x,\Theta}(i_{\text{out}}, l)$	$=$	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}}, i_{\text{out}}, l) \cdot z_{x,\Theta}(i_{\text{in}}, l-1)$
GV	:	$G_{x,\Theta}(i_{\text{out}}, l)$	$=$	$\mathbf{1}_{\{(q_{x,\Theta}(i_{\text{out}}, l)) > 0\}}$
HUO	:	$z_{x,\Theta}(i_{\text{out}}, l)$	$=$	$q_{x,\Theta}(i_{\text{out}}, l) \cdot G_{x,\Theta}(i_{\text{out}}, l)$
FO	:	$\hat{y}_{\Theta}(x)$	$=$	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}}, i_{\text{out}}, d) \cdot z_{x,\Theta}(i_{\text{in}}, d-1)$

Table 4: Here IL, PA, GV, HUO, GL and FO are abbreviations for input layer, pre-activation, gating values, hidden unit output, GAP-layer and final output respectively.

(ii) $rot(x, r)(i) = x(i \oplus r), i \in [d_{in}]$.

(iii) $q_{x,\Theta}(i_{fout}, i_{out}, l) = \sum_{i_{cv}, i_{in}} \Theta(i_{cv}, i_{in}, i_{out}, l) \cdot z_{x,\Theta}(i_{fout} \oplus (i_{cv} - 1), i_{in}, l-1)$.

Definition C.2 (Pooling)**.** *Let* $G^{pool}_{x,\Theta}(i_{fout}, i_{out}, d_{cv} + 1)$ *denote the pooling mask, then we have*

$$z_{x,\Theta}(i_{out}, d_{cv} + 1) = \sum_{i_{fout}} z_{x,\Theta}(i_{fout}, i_{out}, d_{cv}) \cdot G^{pool}_{x,\Theta}(i_{fout}, i_{out}, d_{cv} + 1),$$

where in the case of global-average-pooling $G^{pool}_{x,\Theta}(i_{fout}, i_{out}, d_{cv} + 1) = \frac{1}{d_{in}}, \forall i_{out} \in [w], i_{fout} \in [d_{in}]$, *and in the case of* max-pooling*, for a given* $i_{out} \in [w]$, $G^{pool}_{x,\Theta}(i_{\max}, i_{out}, d_{cv} + 1) = 1$ *where* $i_{\max} = \arg\max_{i_{fout}} z_{x,\Theta}(i_{fout}, i_{out}, d_{cv})$*, and* $G^{pool}_{x,\Theta}(i_{fout}, i_{out}, d_{cv} + 1) = 0, \forall i_{fout} \neq i_{\max}$.

Proposition C.1. *The total number of paths in a CNN is given by* $P^{cnn} = d_{in}(w_{cv} w)^{d_{cv}} w^{(d_{fc}-1)}$.

Notation C.1 (Index Maps)**.** *The ranges of index maps* $\mathcal{I}^f_l$, $\mathcal{I}^{cv}_l$, $\mathcal{I}_l$ *are* $[d_{in}]$, $[w_{cv}]$ *and* $[w]$ *respectively.*

Definition C.3 (Bundle Paths of Sharing Weights)**.** *Let* $\hat{P}^{cnn} = \frac{P^{cnn}}{d_{in}}$*, and* $\{B_1, \ldots, B_{\hat{P}^{cnn}}\}$ *be a collection of sets such that* $\forall i, j \in [\hat{P}^{cnn}], i \neq j$ *we have* $B_i \cap B_j = \emptyset$ *and* $\cup_{i=1}^{\hat{P}^{cnn}} B_i = [P^{cnn}]$*. Further, if paths* $p, p' \in B_i$*, then* $\mathcal{I}^{cv}_l(p) = \mathcal{I}^{cv}_l(p'), \forall l = 1, \ldots, d_{cv}$ *and* $\mathcal{I}_l(p) = \mathcal{I}_l(p'), \forall l = 0, \ldots, d_{cv}$.

Proposition C.2. *There are exactly* d_{in} *paths in a bundle.*

Definition C.4. *Let* $x \in \mathbb{R}^{d_{in}}$ *be the input to the CNN. For this input,*

$$\begin{aligned} A_\Theta(x, p) &\triangleq \left(\Pi_{l=1}^{d_{cv}+1} G_{x,\Theta}(\mathcal{I}^f_l(p), \mathcal{I}_l(p), l)\right) \cdot \left(\Pi_{l=d_{cv}+2}^{d_{cv}+d_{fc}+1} G_{x,\Theta}(\mathcal{I}_l(p), l)\right) \\ \phi_{x,\Theta}(\hat{p}) &\triangleq \sum_{\hat{p} \in B_{\hat{p}}} x(\mathcal{I}^f_0(p)) A_\Theta(x, p) \\ v_\Theta(B_{\hat{p}}) &\triangleq \left(\Pi_{l=1}^{d_{cv}} \Theta(\mathcal{I}^{cv}_l(p), \mathcal{I}_{l-1}(p), \mathcal{I}_l(p), l)\right) \cdot \left(\Pi_{l=d_{cv}+2}^{d_{cv}+d_{fc}+1} \Theta(\mathcal{I}_{l-1}(p), \mathcal{I}_l(p), l)\right) \end{aligned}$$

NPF	$\phi_{x,\Theta} \triangleq (\phi_{x,\Theta}(B_{\hat{p}}), \hat{p} \in [\hat{P}^{cnn}]) \in \mathbb{R}^{\hat{P}^{cnn}}$
NPV	$v_\Theta \triangleq (v_\Theta(B_{\hat{p}}), \hat{p} \in [\hat{P}^{cnn}]) \in \mathbb{R}^{\hat{P}^{cnn}}$

Lemma C.1 (Rotational Invariant Kernel)**.** *Let* H^{cnn}_Θ *denote the NPK of a CNN, then*

$$\begin{aligned} H^{cnn}_\Theta(x, x') &= \sum_{r=0}^{d_{in}-1} \langle x, rot(x', r) \rangle_{\Lambda(\cdot, x, rot(x', r))} \\ &= \sum_{r=0}^{d_{in}-1} \langle rot(x, r), x' \rangle_{\Lambda(\cdot, rot(x, r), x')} \end{aligned}$$

Proof. For the CNN architecture considered in this paper, each bundle has exactly d_{in} number of paths, each one corresponding to a distinct input node. For a bundle $b_{\hat{p}}$, let $b_{\hat{p}}(i), i \in [d_{\text{in}}]$ denote the

path starting from input node i.

$$
\begin{aligned}
&\sum_{\hat{p}\in[\hat{P}]}\left(\sum_{i,i'\in[d_{\text{in}}]} x(i)x'(i')A_\Theta\left(x,b_{\hat{p}}(i)\right)A_\Theta\left(x',b_{\hat{p}}(i')\right)\right)\\
&=\sum_{\hat{p}\in[\hat{P}]}\left(\sum_{i\in[d_{\text{in}}],i'=i\oplus r,r\in\{0,\ldots,d_{\text{in}}-1\}} x(i)x'(i\oplus r)A_\Theta\left(x,b_{\hat{p}}(i)\right)A_\Theta\left(x',b_{\hat{p}}(i\oplus r)\right)\right)\\
&=\sum_{\hat{p}\in[\hat{P}]}\left(\sum_{i\in[d_{\text{in}}],r\in\{0,\ldots,d_{\text{in}}-1\}} x(i)rot(x',r)(i)A_\Theta\left(x,b_{\hat{p}}(i)\right)A_\Theta\left(rot(x',r),b_{\hat{p}}(i)\right)\right)\\
&=\sum_{r=0}^{d_{\text{in}}-1}\left(\sum_{i\in[d_{\text{in}}]} x(i)rot(x',r)(i)\sum_{\hat{p}\in[\hat{P}]}A_\Theta\left(x,b_{\hat{p}}(i)\right)A_\Theta\left(rot(x',r),b_{\hat{p}}(i)\right)\right)\\
&=\sum_{r=0}^{d_{\text{in}}-1}\left(\sum_{i\in[d_{\text{in}}]} x(i)rot(x',r)(i)\Lambda_\Theta(i,x,rot(x',r))\right)\\
&=\sum_{r=0}^{d_{\text{in}}-1}\langle x,rot(x',r)\rangle_{\Lambda_\Theta(\cdot,x,rot(x',r))}
\end{aligned}
$$

□

In what follows we re-state Theorem 3.2.

Theorem C.1. *Let $\sigma_{cv}=\frac{c_{scale}}{\sqrt{ww_{cv}}}$ for the convolutional layers and $\sigma_{fc}=\frac{c_{scale}}{\sqrt{w}}$ for FC layers. Under Assumption 3.1, as $w\to\infty$, with $\beta_{cv}=\left(d_{cv}\sigma_{cv}^{2(d_{cv}-1)}\sigma_{fc}^{2d_{fc}}+d_{fc}\sigma_{cv}^{2d_{cv}}\sigma_{fc}^{2(d_{fc}-1)}\right)$ we have:*

$$
\begin{aligned}
K^v_{\Theta_0^{DGN}} &\to \frac{\beta_{cv}}{{d_{in}}^2}\cdot H^{cnn}_{\Theta_0^f},\ \textit{(for global-average-pooling)}\\
K^v_{\Theta_0^{DGN}} &\to \beta_{cv}\cdot H^{cnn}_{\Theta_0^f},\ \textit{(for global-max-pooling)}
\end{aligned}
$$

Proof. Follows from Theorem 5.1 in [15]. □

D Residual Networks with Skip connections

Lemma D.1 (Sum of Product Kernel)**.** *Let H^{res}_Θ be the NPK of the ResNet, and $H^{\mathcal{J}}_\Theta$ be the NPK of the sub-FC-DNN within the ResNet obtained by ignoring those skip connections in the set $\mathcal{J}$. Then,*

$$H^{res}_\Theta=\sum_{\mathcal{J}\in 2^{[b]}} H^{\mathcal{J}}_\Theta$$

Proof. Proof is complete by noting that the NPF of the ResNet is a concatenation of the NPFs of the 2^b distinct sub-FC-DNNs within the ResNet architecture. □

We re-state Theorem 3.3

Theorem D.1. *Let $\sigma=\frac{c_{scale}}{\sqrt{w}}$. Under Assumption 3.1, as $w\to\infty$, for $\beta^{\mathcal{J}}_{res}=(|\mathcal{J}|+2)\cdot d_{blk}\cdot\sigma^{2\left((|\mathcal{J}|+2)d_{blk}-1\right)}$,*

$$K^v_{\Theta_0^{DGN}}\to\sum_{\mathcal{J}\in 2^{[b]}}\beta^{\mathcal{J}}_{res}H^{\mathcal{J}}_{\Theta_0^f}$$

Proof. Follows from Theorem 5.1 in [15]. □