Duality simplifies deep neural networks with rectified linear units

Anonymous Author(s)
Affiliation
Address
email

Abstract

Rectified linear unit (ReLU) is also a gate which either blocks (multiplies by 0) or allows (multiplies by 1) its pre-activation input. Recently Lakshminarayanan and Singh [2020] showed that in deep neural networks (DNNs) with ReLU (i) most information is in the gates, (ii) learning in the gates during training explains why finite width DNNs outperform their corresponding infinite width *neural tangent kernel* (NTK) (iii) gates can be decoupled from the weights to simplify the NTK into a *neural path kernel* (NPK) which is only dependent on the input and the gates.

In this paper, we explicitise the role of gates, width and depth by showing that $\text{NPK}(x,x') = \langle x, x'\rangle \cdot \Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x')\rangle}{w}$, where x, x' is a pair of inputs, $\langle \cdot, \cdot \rangle$ denotes the inner product, w is width, d is depth and $G_l(x) \in \{0,1\}^w$ is the binary feature vector which encodes the gates in layer l for input x. We also show that in the presence of convolutions with pooling, the NPK is rotationally invariant, and in the presence of skip connections the NPK has a sum of product of base kernels structure. Based on our theory we setup novel experiments that uncover several perviously unknown properties of DNNs with ReLU $-$ the experiments add more evidence to the claims that *most information is in the gates and gates are learnt*.

1 Introduction

Recent works have connected the training and generalisation of deep neural networks (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:

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

$\hat{y}_\Theta(\cdot) \in \mathbb{R}$ 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 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.

Duality for DNNs with Rectified Linear Units (ReLUs). Lakshminarayanan and Singh [2020], [9] observed that each ReLU is also a gate which either blocks (multiplies by 0) or allows its pre-activation input (multiplies by 1) and showed that in a DNN with ReLUs most useful information is learnt its gates. To this end, they used *dual view* to analytically and empirically characterise the role of gating. 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 processing 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'. The output is the sum of the contribution of the individual paths.

• **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 (see Section 2.3). Now, the information in the gates is be measured by fixing the gates, training only the weights and looking at the test performance.

• **Gates = Features:** It was shown that 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 stored in the gates**. Further, gates of a trained network perform better than NTK$_\infty$ and gates from a untrained network performs poorly than NTK$_\infty$, 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 through such gates. This active sub-network is responsible for producing the output for that input. When the gates are decoupled from the weights, the NTK simplifies into a constant times a *neural path kernel* (NPK) given by :

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

where $\textbf{overlap}(x,x')$ is a correlation matrix that measure 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

The dual view [9] empirically showed that information is stored in the active sub-networks and provided an analytical characterisation of the it in terms of the NPK (1). In this paper, we extend the dual view to provide newer insights about the gates. We list the novel contributions in this paper. We 're-write' (1) (a straightforward step) in a manner that explicitises the role of gates, depth and width. In fully connected/dense case with depth 'd' (number of layers) and width 'w' (number of hidden units in each layer), we it follows that

$$\text{NTK}(x,x') \propto \langle x,x'\rangle \cdot \Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x')\rangle}{w}, \tag{2}$$

where $G_l(x) \in \{0,1\}^w$ is the binary feature encoding the gates of layer l. This provides the following hierarchical interpretation: the basic building units are the gates which form the binary feature $G_l(x)$ of that layer; width gives averaging (division by w) ; depth provides product structure $\Pi_{l=1}^{d-1}(\cdot)$. We show via experiments the following properties :

1. Gates are robust to layers permutations, and arbitrary tiling cum rotation.
2. Gates are robust to training and testing with a constant input.
3. Gates are resist against to training first with random labels and then with true labels.
4. Feature extraction can be completely linear and separate from gating and the weights.

Technical Extension (Appendix). In addition, we also extend the dual view to cover the cases of convolutions with pooling and skip connections. We show that convolutional layers with pooling makes the NPK *rotationally invariant* (this property has already been noted to hold for the NTK). We also consider a residual network (ResNet) architecture with $(b+2)$ blocks of fully connected networks with 'b' skip connections between them. Within this ResNet there are $i = 1,\dots,2^b$ possible fully connect networks, and then $\text{NPK}^{\text{residual}} = \sum_{i=1}^{2^b} C_i \text{NPK}_i^{\text{dense}}$, where $C_i > 0$ are positive constants.

2 Prior Work on Duality: Neural Path Framework

In this section, we will present a brief overview of dual view for fully connected DNN (FC-DNN) with ReLU as presented in [9].

2.1 Input Output Relationship in Fully Connected DNN: Primal View

We consider fully-connected DNNs with 'w' hidden units per layer and '$d-1$' hidden layers. $\Theta \in \mathbb{R}^{d_{\text{net}}}$ are the network weights, where $d_{\text{net}} = d_{\text{in}}w + (d-2)w^2 + w$. The information flow is shown in Table 1, where $\Theta(i,j,l)$ is the weight connecting the j^{th} hidden unit of layer $l-1$ to the i^{th} hidden unit of layer l. Further, $\Theta(\cdot,\cdot,1) \in \mathbb{R}^{w\times d_{\text{in}}}, \Theta(\cdot,\cdot,l) \in \mathbb{R}^{w\times w}, \forall l \in \{2,\dots,d-1\}, \Theta(\cdot,\cdot,d) \in \mathbb{R}^{1\times w}$.

Input Layer	:	$z_{x,\Theta}(0)$	$=$	x
Pre-Activation Input	:	$q_{x,\Theta}(i,l)$	$=$	$\sum_j \Theta(i,j,l)\cdot z_{x,\Theta}(j,l-1)$
Gating Values	:	$G_{x,\Theta}(i,l)$	$=$	$\mathbb{1}_{\{q_{x,\Theta}(i,l)>0\}}$
Hidden Layer Output	:	$z_{x,\Theta}(i,l)$	$=$	$q_{x,\Theta}(i,l)\cdot G_{x,\Theta}(i,l)$
Final Output	:	$\hat{y}_\Theta(x)$	$=$	$\sum_{j\in[w]} \Theta(1,j,d-1)\cdot z_{x,\Theta}(j,d-1)$

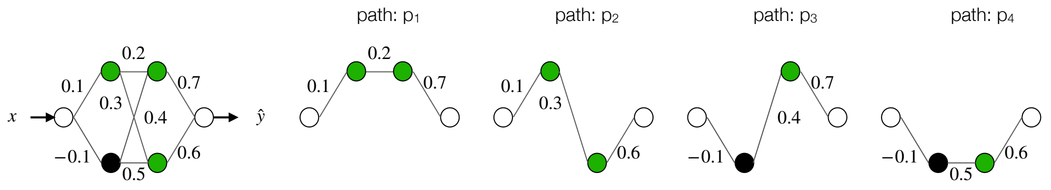
Table 1: Here, $l \in [d-1], i \in [w], j \in [d_{\text{in}}]$ for $l=1$ and $j \in [w]$ for $l = 2,\dots,d-1$.

2.2 Encoding Gates and Weights: Neural Path Feature (NPF) and Neural Path Value (NPV)

Definition 2.1. *A path starts from an input node, passes through a weight and a hidden node 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 $A_\Theta(x,p)$.*

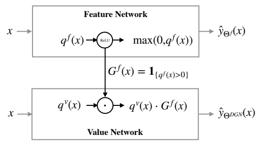
In a FC-DNN with 'd' layers and w hidden units in each layer, there are $P^{\text{fc}} = d_{\text{in}}w^{(d-1)}$ paths. A path is active only if all the gates in the path are active.



Proposition 2.1. *Assuming that the paths can be enumerate as $1,\dots,P^{fc}$, 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.,*

$$\hat{y}_\Theta(x) = \langle \phi_{x,\Theta}, v_\Theta \rangle \tag{3}$$

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



Feature Network	Value Network
$z_{x,\Theta}(0) = x$	$z_{x,\Theta}(0) = x$
$q_{x,\Theta}(i,l) = \sum_j \Theta(i,j,l)\cdot z_{x,\Theta}(j,l-1)$	$q_{x,\Theta}(i,l) = \sum_j \Theta(i,j,l)\cdot z_{x,\Theta}(j,l-1)$
$G_{x,\Theta}(i,l) = \mathbb{1}_{\{q_{x,\Theta}(i,l)>0\}}$	$G_{x,\Theta}(i,l) = \mathbb{1}_{\{q_{x,\Theta}(i,l)>0\}}$
$z_{x,\Theta}(i,l) = q_{x,\Theta}(i,l)\cdot G_{x,\Theta}(i,l)$	$z_{x,\Theta}(i,l) = q_{x,\Theta}(i,l)\cdot G_{x,\Theta}(i,l)$
$\hat{y}_\Theta(x) = \sum_{j\in[w]} \Theta(1,j,d-1)\cdot z_{x,\Theta}(j,d-1)$	$\hat{y}_\Theta(x) = \sum_{j\in[w]} \Theta(1,j,d-1)\cdot z_{x,\Theta}(j,d-1)$

Figure 1: DGN

Note that 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. Hence, in order to understand the roles of $\phi_{x,\Theta}$ and v_Θ it is a good idea to separate them. This is achieved by the deep gated network (DGN) setup (see Figure 1), which has two networks namely the *feature network* parameterised by $\Theta^{\text{f}} \in \mathbb{R}^{d^{\text{f}}_{\text{net}}}$ which holds the NPFs (i.e., the gating information) and a *value network* parameterised by $\Theta^{\text{v}} \in \mathbb{R}^{d^{\text{v}}_{\text{net}}}$ which holds the NPV. The combined parameterisation is denoted by $\Theta^{\text{DGN}} = (\Theta^{\text{f}}, \Theta^{\text{v}}) \in \mathbb{R}^{d^{\text{f}}_{\text{net}} + d^{\text{v}}_{\text{net}}}$.

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

$$K_{\Theta^{DGN}} = K^{v}_{\Theta^{DGN}} + K^{f}_{\Theta^{DGN}}$$

Remark: In the case of fixed regimes, $K^{\mathrm{f}}_{\Theta^{\mathrm{DGN}}} = 0$.

2.4 Overlap of Sub-Networks and Neural Path Kernel

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

overlap$_{\Theta}(i,x,x') = \Lambda_{\Theta}(i,x,x') \overset{\Delta}{=} \left|\{p \in [P] \colon \mathcal{I}^{f}_{0}(p) = i, A_{\Theta}(x,p) = A_{\Theta}(x',p) = 1\}\right|$

Definition 2.3. *Let* $D \in \mathbb{R}^{d_{in}}$ *be a vector of non-negative entries and for* $u, u' \in \mathbb{R}^{d_{in}}$ *, let* $\langle u, u'\rangle_D = \sum_{i=1}^{d_{in}} D(i)u(i)u'(i)$.

Lemma 2.1. *Let* $H_{\Theta}(x,x') \overset{\Delta}{=} \langle \phi_{x,\Theta}, \phi_{x',\Theta}\rangle$ *be the NPK. Then*

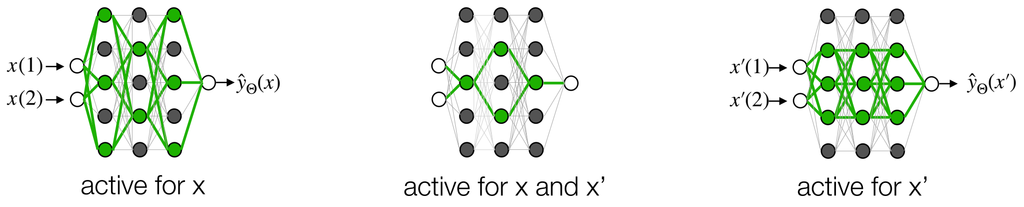
$$H_{\Theta}(x,x') = \langle x, x'\rangle_{\Lambda_{\Theta}(\cdot,x,x')}$$

2.5 NTK $\propto$ NPK

Assumption 2.1. *(i)* Θ^{v}_{0} *is statistically independent of* Θ^{f}_{0} *(ii)* Θ^{v}_{0} *are i.i.d symmetric Bernoulli over* $\{-\sigma, +\sigma\}$.

Theorem 2.1. *Let* $\sigma = \frac{c_{scale}}{\sqrt{w}}$. *Under Assumption 2.1, a* $w \to \infty$, *for FC-DGN we have:*

$$K^{v}_{\Theta^{DGN}_{0}}(x,x') \to d \cdot \sigma^{d-1} \cdot H_{\Theta^{f}_{0}}(x,x') = d \cdot \sigma^{d-1} \cdot \langle x, x'\rangle_{\Lambda_{\Theta^{f}_{0}}(\cdot,x,x')}$$



3 Theoretical Result

Tell them why we are doing this

Theorem 3.1. *Let* $\sigma = \frac{c_{scale}}{\sqrt{w}}$ *and* $\beta_{fc} = d \cdot \sigma^{d-1}$ *Under Assumption 2.1, a* $w \to \infty$, *for FC-DGN we have:*

$$K^{v}_{\Theta^{DGN}_{0}}(x,x') \to \beta_{fc} \cdot \langle x, x'\rangle \cdot \Pi_{l=1}^{d-1} \frac{\langle G_l(x), G_l(x')\rangle}{w}$$

• **Role of Weights:** The weights of the feature network that dictate the NPFs (and hence the NPK) are more critical than the NPV (as long as the NPV is chosen as per Assumption 2.1). Thus, the primary role of the weights is to generate the gates, i.e., the NPFs/NPK. In particular, copying the weights of a pre-trained DNN onto the feature network and training the NPV from scratch recovers the test accuracy of original pre-trained DNN within 1% (see Section 4).

• **Role of Activations:** The ReLU is a special activation, that is, it acts as a gate and the gates themselves are learnt. In particular, learnt gates achieve better test accuracy than random gate (see Section 4). Also note that, Theorem 3.1 applies to any arbitrary gating, i.e., the $w \to \infty$ is only binding on the value network, in the case of feature network with finite width, the gates can be *repeated* infinite times to meet the $w \to \infty$ condition.

• **Width** is responsible for averaging (division by w) of the base kernels.

• **Depth** gives rise to a product of base kernels.

• **Permutation Invariance:** The NPK is invariant if the layers (as masks) are permuted. This is because the $\Pi_{l=1}^{d-1}\frac{H^{\text{lyr}}_{l,\Theta^{\text{f}}_0}(x,x')}{w}$ is permutation invariant. We verify this experimentally (see Section 4).

• **Constant Inputs:** Even if the input to the value network is set of a constant, i.e., even if $\langle x, x'\rangle$ is made constant, the gates still hold information in the from of $\Pi_{l=1}^{d-1}\frac{H^{\text{lyr}}_{l,\Theta^{\text{f}}_0}(x,x')}{w}$. We verify this experimentally (see Section 4).

4 Numerical Experiments

4.1 Experiment 1: DGN with 4 Gating Regimes and 48 Models Per Regime

We make use of the deep gated network (DGN) setup shown in Figure 2 (right) which is an improvisation of the DGN in prior work [9] shown in Figure 2 (left).

Prior Work (4 Regimes, 1 Model Per Regime). Recall from Section 2.3, that a DGN consists of two distinct networks of identical architecture namely the feature network and the value network. Both the feature network and value network have 4 convolutional layers $C1,\dots,C4$ followed by a global average pooling (GAP) layer, a linear layer and a final 'softmax' layer. $\hat{y}_{\text{DGN}}$ is the output of the DGN (performance is measured with respect to this output). $\hat{y}_{\text{f}}$ is used only when we want to *pre-train* the feature network. The feature network has ReLUs, whereas the value network uses the gates from feature network as external masks. Here, $G_1,\dots,G_4$ are the gates of layers $1,\dots,4$ of feature network and in the prior setup these are used directly in the value network. The DGN has the following 4 gating regimes based on the initialisation and trainability of the feature network parameters. In all these regimes, the output of the DGN is $\hat{y}_{\text{DGN}}$.

1. *Fixed Random-Dependent Initialisation* (**FR-DI**): Both value network and feature network are initialised at random and identically, i.e., $\Theta^{\text{f}}_0 = \Theta^{\text{v}}_0$. Only the value network is trained and the feature network is fixed, i.e., $\Theta^{\text{f}}_t = \Theta^{\text{f}}_0, \forall t \geq 0$.

2. *Fixed Random-Independent Initialisation* (**FR-II**): Both value network and feature network are initialised at random and statistically independent of each other, i.e., $\Theta^{\text{f}}_0 \perp \Theta^{\text{v}}_0$. Only the value network is trained and the feature network is fixed, i.e., $\Theta^{\text{f}}_t = \Theta^{\text{f}}_0, \forall t \geq 0$.

3. *Fixed Learnt* (**FL**): The feature network is **pre-trained** first using $\hat{y}_{\text{f}}$ as output. Then the value network is initialised at random, and only the value network is trained and the feature network is fixed.

4. *Decoupled learning* (**DL**): Both value and feature networks are initialised at random, and **both are trained**. In order to ensure that gradient flows through the feature network, we make use of *soft-gating*, i.e., $G(q) = \frac{1}{1+\exp -\beta\cdot q}$, with $\beta = 10$.

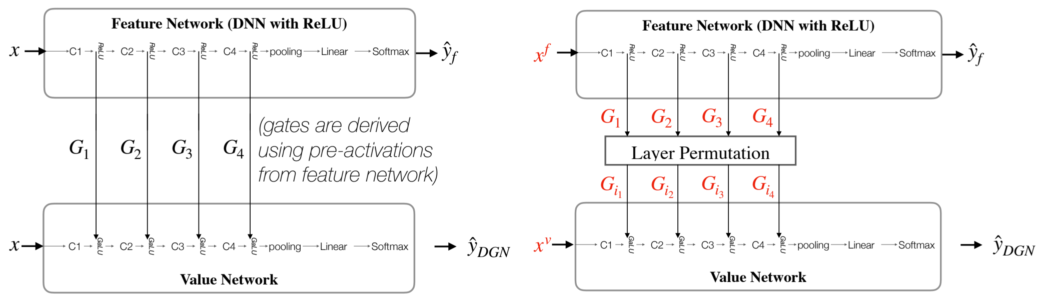


Figure 2: The prior setup in [9] is shown in the left and the setup in this paper is shown in the right. Here $C1,\dots,C4$ are convolutional layers with 128 output filters, kernel size 3×3 and stride 1×1.

This Paper (4 Regimes, 48 Models Per Regime). The improvisations are marked in red. In the current setup, $G_{i_1},\dots,G_{i_4}$ **is a permutation of the** $G_1,\dots,G_4$, this gives 24 models. 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) **constant**: 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 $\mathbf{48} = \mathbf{24} \times \mathbf{2}$ **different models**, where $24 = \texttt{factorial}(4)$ is due to the gate permutations and 2 is due to the two different ways of setting x^{v}.

	FR-II	FR-DI	DL	FL	ReLU
FC (MNIST)	94.1%	94.1%	98.1%	98.6%	98.5%
CNN (CIFAR-10)	67.5%	67.6%	77.6%	79.4%	80.4%

Discussion.

1. **Decoupling gates and weights does not hurt.** There is no performance difference between FR-II and FR-DI. Further, decoupled learning of gates (DL) performs significantly better than fixed random gates (FR), and the gap between standard DNN with ReLU and DL is less than 3%. This marginal performance loss seems to be worthy trade off for fundamental insights of Theorem 3.1 under the decoupling assumption.

2. **Features are in the gates.** The fixed learnt regime (FL) shows that using the gates of a pre-trained ReLU network, performance can be recovered by training the NPV. Also, by interpreting the input dependent component of a model to be the features and the input independent component to be the weights, it makes sense to look at the gates/NPFs as the hidden features and NPV as the weights.

3. **Random gates perform well.** FR-II does perform well in all the experiments (note that for a 10-class problem, a random classifier would achieve only 10% test accuracy). Given the observation that the gates are the true features, and the fact that is there no learning in the gates in the fixed regime, and the performance of fixed random gates can be purely attributed to the in-built structure.

4. **Gate Learning explain finite vs infinite width.** We group the models into three sets where $S_1 = \{$ ReLU, FL , DL$\}$, $S_2 = \{$ FR$\}$ and $S_3 = \{$ CNTK $\}$, and explain the difference in performance due to gate learning. S_2 and S_3 have no gate learning. However, S_3 due to its infinite width has better averaging resulting in a well formed kernel and hence performs better than S_2 which is a finite width. Thus, the difference between S_2 and S_3 can be attributed to finite versus infinite width. Both S_1 and S_2 are finite width, and hence, conventional feature learning happens in both S_1 and S_2, but, S_1 with gate learning is better (77.5% or above in CIFAR-10) than S_2 (67% in CIFAR-10) with no gate learning. Thus neither finite width, nor the conventional feature learning explain the difference between S_1 and S_2. Thus, 'gate learning' discriminates the regimes S_1, S_2 and S_3 better than the conventional feature learning view.

5. **Robustness to permutation:** The performance (in all the 4 regimes) is also robust to permutation of layers.

6. **Robustness to constant input:** The performance (in all the 4 regimes) is robust to 'all-ones' inputs. Note that in the 'all-ones' case, the input information affects the models only via the gates. Here, all the entries of the input Gram matrix are identical, and the NPK depends only on the base kernels.

7. **Are Features Learnt Hierarchically? Yes and No.**

4.2 Experiment 2: Upstream training with random labels and downstream with true labels

Q1 (open question). When trained with random labels upstream followed by true labels downstream, the test performance of DNNs 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. 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 2 (left), and train the feature network (which is a DNN with ReLUs). In 'Phase I: Upstream', we train the feature network for different values of label noise $\gamma = 0, 25\%, 50\%, 75\%$, which gives us models $M1(\gamma)$ (see Figure 4). In order to measure the information in the gates learnt

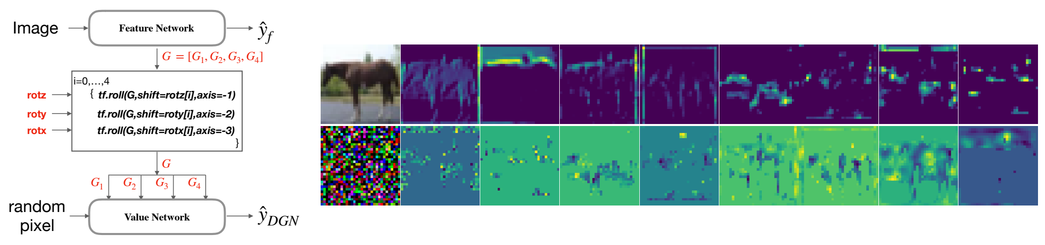


Image
Feature Network
$\hat{y}_f$
$G = [G_1, G_2, G_3, G_4]$
i=0,...,4
rotz
roty
rotx
tf.roll(G,shift=rotz[i],axis=-1)
tf.roll(G,shift=roty[i],axis=-2)
tf.roll(G,shift=rotx[i],axis=-3)
G
G_1 G_2 G_3 G_4
random pixel
Value Network
$\hat{y}_{DGN}$

Figure 3: Top (Standard CNN): First image on the left is the input image and the next 8 images are outputs of 2 filters in each of the 4 layers. Bottom (DGN with gates of the top model applied in reverse order): First image on the left is the input to the value network and the next 8 images are outputs of 2 filters in each of the 4 layers. Both models achieve a test accuracy of about 80%.

at the end of 'Phase I: Upstream', we keep these gates fixed and train the value network with true labels – this gives us models $M2(\gamma)$ (see Figure 4). Second is 'Phase II: Downstream', wherein, we start with models $M1(\gamma)$, and perform downstream training with true labels to obtain models $M4(\gamma)$ ($\gamma = 0$ is an exception; there is no downstream training because $M1(0)$ has been already trained with true labels in upstream). In order to measure the information in the gates learnt at the end of 'Phase II: Downstream', we keep these gates fixed and train the value network with true labels – this gives us models $M5(\gamma)$ (see Figure 4).

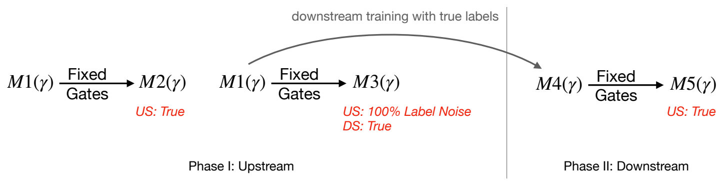


Figure 4: Shows various models trained in Experiment 2. Here, US and DS stand for upstream and down stream respectively. True for with true labels. Here models $M1$ and $M4$ are feature networks (DNNs with ReLUs), and $M2$, $M3$ and $M5$ are value networks which use the fixed gates from the feature networks. Models are parameterised by (γ) which is the label noise level.

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: Upstream', 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 $M3(\gamma)$.

We now discuss the results of the experiments with random labels shown in Table 2 and Figure 5.

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 $M2(\gamma = 0)$ and compare it performance of $M4(\gamma), \gamma = 25, 50.75$. When the gates are fixed, the upstream training with even 100% label noise does not hurt the test performance so much (less than 2% for $\gamma = 100$) in comparison to the cases when the gates are allowed to change (as it is the case of DNN with ReLUs), 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. *When training with random labels upstream, the test performance degrades because the gates get degraded.* This follows from the fact that the performance of the DNN with ReLUs at the end of 'Phase II: Downstream', i.e., $M4$ and that of the value network with the fixed gates at end of 'Phase II: Downstream', i.e., $M5$ are approximately equal (within 0.5%).

Gates are fairly robust to random labels. Note that while training with random labels upstream, the test performance initially improves, reaches a peak and then degrades (see top row of Figure 5). The performance of the DNN with ReLUs at the end of 'Phase I' is shown in the second column of Table 2. However, the performance of the fixed gates at the end of 'Phase I' shown in third column of Table 2 is much better and is more closer to the performance of the downstream model $M4$. This

	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 performance of the various models in Experiment 2.

means, during upstream training with random labels, the performance degradation post the peak value mostly affects only the weights and not the gates.

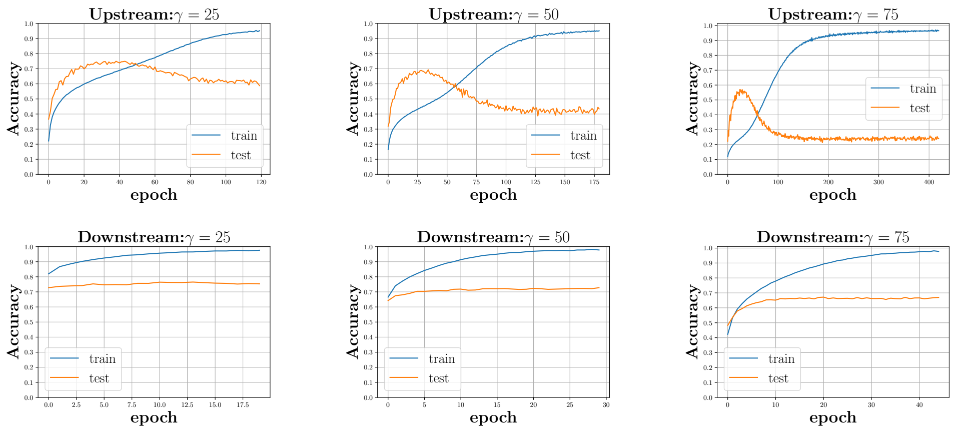
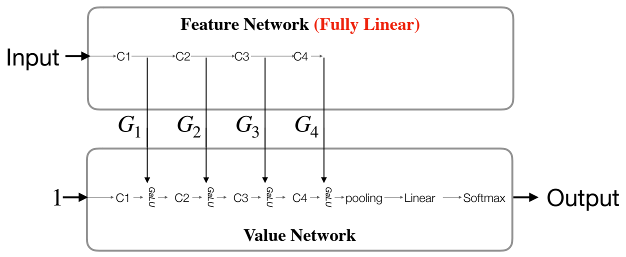


Figure 5: Shows the upstream training with random labels followed downstream training with true labels. The first epochs of bottom plots is same as the epoch follows the last epoch in the top plots.

5 A Simple And Interpretable Deep Gated Network



Feature Network is a fully linear network and comprises of w convolutional windows in each layer. In order to make it simple, let us say tag the convolutional windows based on their ‘image processing’ functionalities such as *edge detection, sharpening, blurring* etc, and let us say that there are K such functionalities. Let us denote these functionalities by operators $\mathcal{F}_1 \ldots, \mathcal{F}_K$. For input $x \in \mathbb{R}^{d_{\text{in}}}$ let us examine the outputs of various layers. Layer 1 output has $\mathcal{F}_1 \circ x, \ldots, F_2 \circ x$, and in layer 2 output is given by $\mathcal{F}_i \circ \mathcal{F}_j x, i, j = 1, \ldots, K$ and layer d contains $\mathcal{F}_{i_d} \ldots \circ \mathcal{F}_{i_1} x, i_1, \ldots, i_d = 1, \ldots, K$. Note that we have used $\circ$ to denote composition of functionalities, however $\mathcal{F}_i \circ \mathcal{F}_j$ in the network it is indeed a multiplication of the weight matrices θ_a and θ_b $(a, b = 1, \ldots, w)$ which have functionalities $\mathcal{F}_i$ and $\mathcal{F}_j$. Thus the feature network can be completely understood in ‘image processing’ functionalities and standard linear algebraic tools.

Gating. Each gate is triggered based on whether or not the layer input aligns with respect to the hyperplane given by the incoming weights of the gate. The gates in a layer gives rise to the binary feature vector in the dimension equal to the number of gating units in that layer.

Value Network. Uses the gates from the feature network, lays them out depth-wise. From **??** we know that in the limit of infinite width value network here implements the rotationally invariant NPK, and from the experimental results that the difference between finite and infinite width is in gate learning.

Training and Testing.

References

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.

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.

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.

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.

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.

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.

Kaixuan Huang, Yuqing Wang, Molei Tao, and Tuo Zhao. Why do deep residual networks generalize better than deep feedforward networks?—a neural tangent kernel perspective. *Advances in Neural Information Processing Systems*, 33, 2020.

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

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.

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.

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.

Yiping Lu, Chao Ma, Yulong Lu, Jianfeng Lu, and Lexing Ying. A mean field analysis of deep resnet and beyond: Towards provably optimization via overparameterization from depth. In *International Conference on Machine Learning*, pages 6426–6436. PMLR, 2020.

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.

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? **[TODO]**
   (b) Did you describe the limitations of your work? **[TODO]**
   (c) Did you discuss any potential negative societal impacts of your work? **[TODO]**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? **[TODO]**
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? **[TODO]**
   (b) Did you include complete proofs of all theoretical results? **[TODO]**
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)? **[TODO]**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? **[TODO]**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? **[TODO]**
   (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)? **[TODO]**
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? **[TODO]**
   (b) Did you mention the license of the assets? **[TODO]**
   (c) Did you include any new assets either in the supplemental material or as a URL? **[TODO]**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? **[TODO]**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **[TODO]**
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? **[TODO]**
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? **[TODO]**
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? **[TODO]**

A Appendix

Optionally include extra information (complete proofs, additional experiments and plots) in the appendix. This section will often be part of the supplemental material.