IDINIT: A UNIVERSAL AND STABLE INITIALIZATION METHOD FOR NEURAL NETWORK TRAINING

Yu Pan[1] **Chaozheng Wang**[2] **Zekai Wu**[3] **Qifan Wang**[4] **Min Zhang**[1] **Zenglin Xu**[56*]
[1]Harbin Institute of Technology, Shenzhen [2]The Chinese University of Hong Kong
[3]The Hong Kong Polytechnic University [4]MetaAI [5]Fudan University
[6]Shanghai Academy of AI for Science
iperryuu@gmail.com czwang23@cse.cuhk.edu.hk zenglin@gmail

ABSTRACT

Deep neural networks have achieved remarkable accomplishments in practice. The success of these networks hinges on effective initialization methods, which are vital for ensuring stable and rapid convergence during training. Recently, initialization methods that maintain identity transition within layers have shown good efficiency in network training. These techniques (e.g., Fixup) set specific weights to zero to achieve identity control. However, settings of remaining weight (e.g., Fixup uses random values to initialize non-zero weights) will affect inductive bias that is achieved only by a zero weight, which may be harmful to training. Addressing this concern, we introduce fully identical initialization (IDInit), a novel method that preserves identity in both the main and sub-stem layers of residual networks. IDInit employs a padded identity-like matrix to overcome rank constraints in non-square weight matrices. Furthermore, we show the convergence problem of an identity matrix can be solved by stochastic gradient descent. Additionally, we enhance the universality of IDInit by processing higher-order weights and addressing dead neuron problems. IDInit is a straightforward yet effective initialization method, with improved convergence, stability, and performance across various settings, including large-scale datasets and deep models.

1 INTRODUCTION

Deep neural networks have attracted significant attention due to their versatility in various applications (He et al., 2016; Li et al., 2021; Wang et al., 2023). Behind these successes, initialization methods play a crucial role in promoting stable and fast-convergent training processes for networks (Sutskever et al., 2013; Arpit et al., 2019; Pan et al., 2022; 2024). Usually, initialization methods make effects by controlling the magnitude of signals. For example, Xavier (Glorot & Bengio, 2010) initialization is originally proposed to maintain signals in the non-saturated region of the sigmoid activation function by restricting signal variances, which greatly solved the difficulty of training. Then, Poole et al. (2016) propose to initialize network weights by constraining signals on the edge of chaos through dynamical isometry, which can further benefit the network training. Later, Hardt & Ma (2017) analyzed the optimization landscape of linear residual networks, and found that weights that transit identity in layers can help networks converge fast as their F-norm is close to that of the final converged weights. And identity transition also corresponds to isometry theory (Zhang et al., 2019), thereby, contributing to avoiding gradient explosion and diffusion.

An instance of preserving identity across neural network layers, known as "identity-control," is depicted in Figure 1 and formally expressed as $Y = X$. This type of initialization can be implemented by setting specific weights (e.g., W_2) to $\mathbf{0}$, thereby ensuring zero output in the sub-stem, as elucidated by Hardt & Ma (2017). This approach, however, poses challenges in configuring the remaining

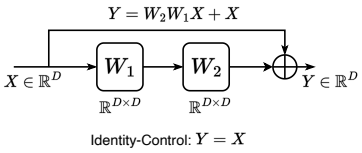
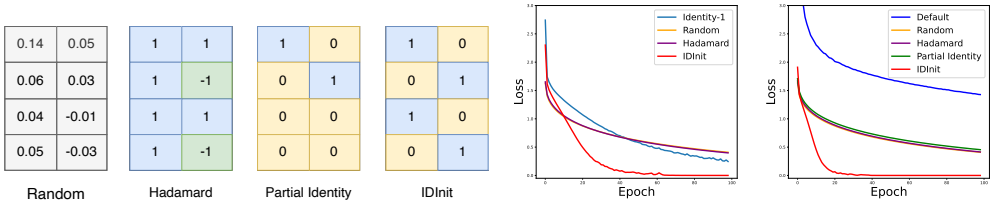


Figure 1: A case of identity-control initialization, which sets $W_2 = \mathbf{0}$ to satisfy $Y = X$.

*Corresponding author.



(a) Initialization methods. (b) Square Loss. (c) Rectangle Loss.

Figure 2: Analyzing effect of initializing W_1 while $W_2 = \mathbf{0}$. The experiment uses Cifar10 and blocks in Figure 1, and more details are in Appendix C.5. (a) The initialization methods for W_1 in a rectangular format. Fixup: "Random"; ZerO: "Hadamard". And "Partial Identity" and "IDInit" denote padding $\mathbf{0}$ and I to an identity matrix, respectively. (b) Set $W_1 \in \mathbb{R}^{240\times240}$ and $W_2 \in \mathbb{R}^{240\times240}$ as square matrices. "Identity-1" represents a configuration where only one weight is initialized as $\mathbf{0}$. Interestingly, while "Random" and "Hadamard" methods may outperform "Identity-1" in initial training epochs due to more network weights, they are hard to capture the inductive bias of "Identity-1", resulting in convergence difficulties. In contrast, IDInit can effectively leverage the training dynamics associated with "Identity-1". (c) Set $W_1 \in \mathbb{R}^{280\times240}$ and $W_2 \in \mathbb{R}^{240\times280}$ as rectangle matrices. "Default" means W_1 and W_2 are initialized with Xavier. However, "Default" proves ineffective for training, as it conflicts with dynamical isometry. Furthermore, even though "Partial Identity" exhibits the capability to transmit partial signals, it performs poorly due to rank constraint issues. Finally, IDInit maintains well-training conditions by padding the identity matrix.

weight W_1. Previous work such as Fixup (Zhang et al., 2019) and ZerO (Zhao et al., 2022) initialize W_1 using the Xavier and Hadamard methods, respectively. These initializations can adversely affect the inductive bias already established by setting $W_2 = \mathbf{0}$, a setting beneficial for training. As evidenced in Figure 2, both Xavier and Hadamard methods cause difficulties in achieving convergence. Observing this, we propose initializing W_1 with an identity matrix I, which retains the inductive bias as $IW_2 \equiv W_2$. Moreover, I also achieves dynamical isometry in the sub-stem layer as discussed by Zhao et al. (2022). Figure 2 demonstrates that using an identity matrix significantly aids in training convergence. Nonetheless, the practical application of an identity matrix faces two primary obstacles. First, an identity matrix requires square-shaped weights, a condition seldom met in practical networks. While a partial identity matrix (by padding $\mathbf{0}$ to an identity matrix) offers a workaround, it leads to rank constraints issues (Zhao et al., 2022) when the output dimension exceeds the input dimension, impairing network generalization. The second obstacle concerns the convergence capability. As Bartlett et al. (2019) pointed out, weights initialized with an identity matrix are difficult to converge to the ground truth, of which eigenvalues contain negative values. This convergence problem is important as it indicates a limited universality of applying an identity matrix as an initialization method.

IDInit. In light of the preceding discussion, we aim to address these two major obstacles. To handle a non-square matrix, we pad a new identity matrix in adjacency to an identity matrix. We theoretically demonstrate this operation can resolve the rank constraint problem. Then, to alleviate the replica problem induced by this padding scheme, we impose a loosening condition on the padded identity-like matrix. Turning to the matter of convergence, we conduct an experiment to analyze it. Interestingly, we find that the convergence problem can be solved by adding a moment in an optimizer (e.g., the stochastic gradient descent optimizer), which is the most general setting for training neural networks. By introducing the identity-like matrix into the identity-control framework, we implement a fully identical initialization (IDInit), which ensures identity transition across both main and sub-stem layers. Moreover, we explore two additional techniques for improving the universality of IDInit and the identity-control framework:

(1) Higher-order Weights: An identity matrix is a 2-D array and it is necessary to consider an efficient method to transfer the identity matrix to a higher-order weight (e.g., a 4-D convolution). A previous strategy is to keep identity along the channel (see Sec. 3.1.3). However, this causes diversity loss in channels, which is harmful to performance. To remedy this shortage, we keep identity in patches alternatively for more diversity in channels to achieve improvement.

(2) Dead Neurons: As an identity-control method, IDInit sets the last layer of the sub-stem to 0 for transiting identity in the main branch. However, a dead neuron problem is possibly caused by

this setting, especially for residual convolutional networks (Zhang et al., 2019; Zhao et al., 2022). Addressing this, we select some elements to a small numerical value ε to increase trainable neurons as in Figure 3.

To our knowledge, IDInit is the first successful trial to maintain identity in both main- and sub-stems by breaking the rank constraints, which promise the expressive power of IDInit. Then, we address the replica problem by adding small noise while maintaining the dynamical isometry. By further proposing modifications to CNNs and solutions to dead neuron problems, we have significantly improved accuracy of classifying Cifar10 on ResNet-20 by 3.42% and 5.89%, respectively. (see Section 4.2). Note that, although the identity matrix is used as initialization in prior work, it was only used for square matrix, e.g., Le et al. (2015) set a hidden-to-hidden layer in a recurrent neural network with an identity matrix for better performance. IDInit is novel for the consideration of non-standard situations, e.g., non-square matrix. On ImageNet, compared to the default random initialization, IDInit demonstrates superior performance, achieving an average improvement of 0.55%, and facilitates faster convergence across various settings, reducing the required training time by an average of 7.4 epochs. IDInit can accelerate the training procedure of BERT-Base, manifesting an 11.3% reduction in computational cost. Therefore, our approach yields consistently significant advantages in the training of neural networks.

2 Related Work

Consider an L-layer residual network, each residual block of which consists of a residual connection and a residual stem that refers to the component excluding the residual connection. Assumming each residual stem contains two parameters, and the network's input signal is denoted as $x^{(0)}$, the i-th layer can be formulated as

$$x^{(i+1)} = a(I + \theta^{(i,0)}\theta^{(i,1)})x^{(i)}, \tag{1}$$

where $a(\cdot)$ denotes the activation function, $x^{(i)}$ means an input of i-th residual block in a network, I is an identity matrix denoting residual connection, and $\theta^{(i,0)}$ and $\theta^{(i,1)}$ are weights in the i-th residual stem of a residual block.

Dynamical Isometry. Assuming signal magnitude (e.g., $\sigma^2(x^{(i)})$) of each layer changing in a scale α, the last signal magnitude can reach α^L (e.g., $\sigma^2(x^{(L)}) = \alpha^L\sigma^2(x^{(0)})$), making it easy to cause signal explosion and diffusion, especially for large L. To mitigate this issue, dynamic isometry provides an effective solution. Considering the input-output Jacobian which is defined as

$$J_{io} = \frac{\partial x^{(L)}}{\partial x^{(0)}}, \tag{2}$$

the dynamical isometry is achieved when all the singular values of J_{io} are close to 1. Moreover, with the mean squared singular value of J_{io} noted as χ, Pennington et al. (2017) and Bachlechner et al. (2021) show that $\chi > 1$ indicates that the model is in a chaotic phase, and back-propagated gradients will explode exponentially. By contrast, $\chi < 1$ means a model in an ordered manner that back-propagated gradients vanish exponentially. $\chi = 1$ is a critical line of initialization, avoiding gradient vanishing or exploding. The isometry can provide sufficient robustness for the network training (Gilboa et al., 2019; Poole et al., 2016; Yang & Schoenholz, 2017).

Network Initialization. Common initialization methods are Xavier (Glorot & Bengio, 2010) and Kaiming initialization (He et al., 2015). Especially for residual networks efficiency, Hardt & Ma (2017) theoretically demonstrates that network training benefits from keeping identity. Le et al. (2015) set a hidden-to-hidden layer in a recurrent neural network with an identity matrix for better performance. Fixup (Zhang et al., 2019) and ZerO (Zhao et al., 2022) successfully initialize ResNets by setting residual stem to 0 (not residual connections) to guarantee the identity of signals. SkipInit (De & Smith, 2020) replaces Batch Normalization with a multiplier whose value is 0. ReZero (Bachlechner et al., 2021) directly adds extra parameters of value 0 to keep identity, leading to fast convergence.

Identity-Control Training Framework. Net2Net (Chen et al., 2016) proposes to expand network depth by maintaining identity. DiracNet (Zagoruyko & Komodakis, 2017) maintains an identity

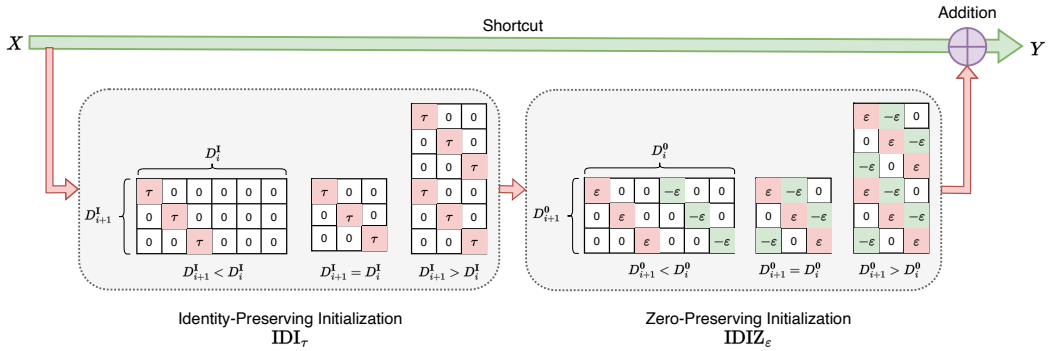


Figure 3: An overview of IDInit, which consists of identity-preserving initialization IDI$_\tau$ and zero-preserving initialization IDIZ$_\varepsilon$, of which dimensions are denoted as $D^{\mathbf{I}}$ and $D^{\mathbf{0}}$. τ and ϵ are usually set to 1 and 1e-6 to maintain identity and transit zero. i and $i+1$ mean two adjacent layer indices.

for propagating information deeper into the network. However, it suffers from reducing residual connection, causing performance loss. ISONet (Qi et al., 2020) is an isometric learning framework that contains an identical initialization (i.e., the Dirac function that is also used in ZerO (Zhao et al., 2022) by padding 0 in a non-square matrix case), and isometric regulation in training. ISONet multiplies 0 to the residual stem like Fixup (Zhang et al., 2019). ISONet lacks the flexibility for various convolutions as it specifies the net without normalization, and requires SReLU.

3 Fully Identical Initialization

The identity-control scheme serves as a practical initialization framework, with prior studies such as Fixup and ZerO demonstrating success within this paradigm. As depicted in Figure 3, IDInit achieves this scheme with two components: identity-preserving initialization and zero-preserving initialization, aimed at transferring identity and zero, respectively. We elaborate on the identity-preserving initialization, which involves padding identity matrices, in Section 3.1, and discuss the zero-preserving initialization, which addresses dead neurons, in Section 3.2.

3.1 Preserving Identity by Padding Identity

A standard identity matrix can naturally satisfy identity transition. However, in a non-square situation, this natural advantage is lost. To address this problem, we pad the identity matrix on an identity matrix to fit a non-square matrix. Specifically, for a fully-connected layer transformed from Eq. (1) as $x^{(i+1)} = \theta^{(i)} x^{(i)}$, we set the weight $\theta^{(i)} \in \mathbb{R}^{D^{\mathbf{I}}_{i+1} \times D^{\mathbf{I}}_i}$ to

$$\theta^{(i)}_{m,j} = \begin{cases} \tau, & \text{if } m \equiv j \pmod{D^{\mathbf{I}}_i}, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The initialization formulated as Eq. (3) is termed as IDI$_\tau$, where IDI means the identical initialization function, and τ is calculated by considering the activation function, e.g., $\tau_{ReLU} = \sqrt{2}$ for the ReLU function. As shown in Figure 3, setting $\tau = 1$ can form IDI$_1$ initialization.

3.1.1 Analysis on Convergence Ability of the Identity Matrix

As proposed by Bartlett et al. (2019), weights initialized with an identity matrix face difficulty in converging towards the target when its eigenvalues include negative values. This implies a potential constraint on the convergence efficacy of the IDInit method. Consequently, we will delve deeper into this issue in the following discussion. According to their study, when layers in a neural network are initialized using the identity matrix, all the weight matrix of layers will be symmetric at each step of the training process. This persistent symmetry leads to the weights of layers always being the same at any training step, causing the aforementioned convergence difficulty. Interestingly, we find that this problem is mainly caused by the gradient descent (GD) which uses all the data in one batch, and employing a stochastic gradient descent (SGD) of which data in different batches can be different,

can effectively break the symmetry in gradients which facilitates convergence, and incorporating momentum can further accelerate the convergence process.

To elaborate on this problem, we present a training case for a single-layer network expressed as $y = \theta x$, where $x \in \mathbb{R}^d$ represents the input, $y \in \mathbb{R}^d$ denotes the output, and $\theta \in \mathbb{R}^{d \times d}$ is the weight matrix. The weight matrix θ is initialized to the identity matrix I, denoted as $\theta^{(0)} = I$. For our loss function, we employ the Mean Squared Error (MSE) and a learning rate denoted by η. Consider two training pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$ sampled from the same dataset $\mathcal{D}$. The network is initially trained with $\{x_1, y_1\}$, and trained with $\{x_2, y_2\}$ in the next step.

Being updated after two steps, the final gradient $\Delta\theta^{(1)}$ can be calculated as

$$x_2 x_2^T - \eta x_1 x_1^T x_2 x_2^T + \eta y_1 x_1^T x_2 x_2^T - y_2 x_2^T. \tag{4}$$

While $x_2 x_2^T$ is symmetric, $x_1 x_1^T x_2 x_2^T$, $y_1 x_1^T x_2 x_2^T$, and $y_2 x_2^T$ can be asymmetric. To quantify the magnitude of the asymmetry in $\Delta\theta^{(1)}$, let $\Omega = -\eta x_1 x_1^T x_2 x_2^T + \eta y_1 x_1^T x_2 x_2^T - y_2 x_2^T$ denote the asymmetric component. The magnitude of the asymmetry can be calculated as $\mathbb{E}(||\Omega - \Omega^T||_F^2)$. Assuming $x_1, x_2, y_1, y_2 \in \mathbb{R}^d$ are random vectors with entries that are i.i.d. Gaussian random variables distributed as $N(0, \sigma^2)$, the magnitude of the asymmetry is bounded as

$$4\eta^2 d^3 \sigma^8 - 4\eta^2 d^2 \sigma^8 + 2d^2 \sigma^4 \leq \mathbb{E}(||\Omega - \Omega^T||_F^2) \leq 6\eta^2 d^3 \sigma^8 + 3d^2 \sigma^4. \tag{5}$$

As η is usually $1e-1$, and both training pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$ can be generally normalized to $\mathcal{N} \sim (0, 1)$, thereby, the symmetry of the weight can be sufficiently influenced as

$$\theta^{(2)} = \theta^{(1)} - \eta \Delta\theta^{(1)}. \tag{6}$$

When introducing a momentum $m^{(0)}$ initialized to $\Delta\theta^{(0)}$, $\theta^{(2)}$ will be updated as

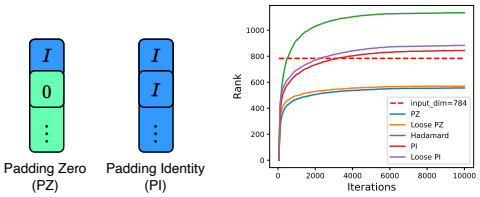
$$\begin{aligned} m^{(1)} &= \gamma m^{(0)} + \eta \Delta\theta^{(1)}, \\ \theta^{(2)} &= \theta^{(1)} - m^{(1)} = \theta^{(1)} - \gamma m^{(0)} - \eta \Delta\theta^{(1)}, \end{aligned} \tag{7}$$

where γ is the coefficient of m. Therefore, momentum can promote the weight to become asymmetric by accumulating the asymmetry of gradients in steps and impact more when samples are increased. We show that SGD with momentum can effectively resolve the issue of layers being the same in networks initialized with the identity matrix during training, which facilitates the convergence process. The completed derivation is provided in Sec. A.2 of the appendix.

As for networks of multiple layers, when their layers are asymmetric, each layer can be updated differently which breaks the convergence problem caused by the same gradients in each step (which is stated in Lemma 5 of Bartlett et al. (2019)). As illustrated in Figure 10 of the appendix, it is evident that layers trained using SGD are different from each other, with the momentum component amplifying the degree of this difference.

3.1.2 On Rank Constraint Problem

ZerO (Zhao et al., 2022) identifies that a dimension-increasing matrix may face a rank constraint problem if padding zero values. In this analysis, we investigate whether padding with an identity matrix results in this constraint.



(a) Padding schemes. (b) Rank plot.

Figure 4: Two padding schemes and their influence on ranks of a layer. We trained a 3-layer network on MNIST, and set $D_0 = 768$ and $D_h = 2048$. We plot $\text{rank}(\Delta\theta^{(1)}) \in \mathbb{R}^{D_h \times D_h}$ in (b). As shown in (b), padding identity can achieve more than a rank of 768 like Hadamard, while padding zero is limited under 768. The loose condition can lead to better rank performance, however, cannot solve the rank constraint problem of padding zero.

Rank Constraint Problem. Consider a 3-layer network with weights $\{\theta^{(i)}\}_{i=0}^2$, where $\theta^{(0)} \in \mathbb{R}^{D_h \times D_0}$, $\theta^{(1)} \in \mathbb{R}^{D_h \times D_h}$, $\theta^{(2)} \in \mathbb{R}^{D_L \times D_h}$ where $D_h > D_0, D_L$. Given an input batch $x^{(0)} \in \mathbb{R}^{D_0 \times N}$ with a size N, the formulation of the i-th layer is $x^{(i+1)} = \theta^{(i)} x^{(i)}$, where $i \in [2]$. Define residual component $\Delta\theta^{(1)} = \theta^{(1)} - I$. When initializing the dimension-increasing weight $\theta^{(0)}$ by padding zeros (PZ) values, the rank constraint problem refers to

$$\text{rank}(\Delta\theta^{(1)}) \leq D_0. \tag{8}$$

This rank constraint issue signifies a performance limitation associated with the initialization method. Intriguingly, our findings indicate that the initialization method IDI_τ successfully avoids this rank constraint, as detailed in Theorem 3.1. The proof is deferred to Appendix A.4.

Theorem 3.1. *If initializing all weights $\{\theta^{(i)}\}_{i=0}^{2}$ by IDI_1, the rank of $\Delta\theta^{(1)}$ can attain*

$$\textit{rank}(\Delta\theta^{(1)}) \geq D_0, \tag{9}$$

which breaks the rank constraint.

Notably, Theorem 3.1 suggests that an IDInit initialized network can break this constraint through SGD without the help of non-linearity like ReLU which is mentioned as necessary in the prior study (Zhao et al., 2022). Specifically, when non-linearity like ReLU is not applied, the rank of the middle weight being limited to D_0 only happens at the beginning. After training for several steps, an IDInit-initialized network can break this constraint.

Replica Problem. When recurrently padding the identity matrix, the output features are still replicated. According to Blumenfeld et al. (2020), such a replica problem can be solved by adding noise to weights. Inspired by that, we loosen the identity condition to generate $\tau \sim N(\tau, \epsilon_\tau)$, while keeping most identity. ϵ_τ is a small value and set to 1e-6 in this paper. With this loose condition, IDInit can give additional noise to output features and bring more feature diversity. Profiting from the feature diversity, IDInit therefore can increase the rank values as shown in Figure 4(b).

3.1.3 Patch-Maintain Convolution

Convolution layers are important structures in deep neural networks. Here, we will explore an initialization pattern for convolution with the identity transition. A convolution kernel is usually defined as $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{k\times k\times c_{in}\times c_{out}}$, where c_{in} and c_{out} denote the number of channels of input and output, respectively, and k denotes convolutional kernel size. Similar to an identity matrix, Zhao et al. (2022) propose a channel-maintain convolution layer that transits identity by setting 0-filled $\boldsymbol{\mathcal{C}}$ through $\mathrm{IDI}_\tau(\boldsymbol{\mathcal{C}}_{n,n,:,:})$, where $n \in \mathbb{N}^+$ and $k = 2n+1$. As a convolutional kernel window size, k is usually an odd number. When $c_{in} = c_{out}$, the convolution maintains the identity. When $c_{in} > c_{out}$ or $c_{in} < c_{out}$, $\boldsymbol{\mathcal{C}}$ will under-sample and over-sample on an input feature along channel respectively. Keeping identity is usually considered as an efficient way to improve model performance, however, we find that this setting can lead to a fatal performance degeneration (see Sec. 4.2).

Patch-Maintain Convolution. Inspired by Han et al. (2020) that enhance model performance by increasing channel diversity, we propose to fuse spatial information by simply reshaping a matrix initialized with IDI_τ. Specifically, we reshape the convolutional kernel $\boldsymbol{\mathcal{C}}$ into a matrix $C \in \mathbb{R}^{c_{out}\times kkc_{in}}$. We initialize C as

$$\mathrm{IDI}_\tau(C). \tag{10}$$

Then by reshaping C into $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{k\times k\times c_{in}\times c_{out}}$, our initialization for a convolution is completed. This reshaping strategy can shift spatial features, thereby increasing feature diversity. We utilize IDIC_τ to denote such a reshaping process. A detailed description is in Figure 11 in the Appendix.

3.2 Preserving Zero by Tackling Dead Neurons

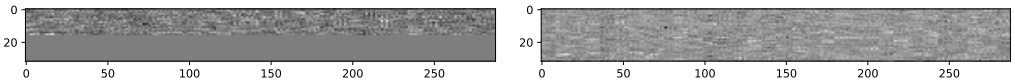
Given a residual network formulated by Eq. (1), prior identity-control initialization (Zhang et al., 2019; Zhao et al., 2022) set the last transformation in the residual stem to 0, i.e., $\theta^{(i,0)} = 0$, thereby maintaining an identity as

$$x^{(i+1)} = (I + 0)x^{(i)} = x^{(i)}. \tag{11}$$

However, the setting can possibly cause dead neurons.

Dead Neuron Problem. The dead neuron problem occurs when a neuron's weight becomes zero and receives zero gradients, rendering it incapable of updating. This issue is harmful to the training performance of models. Fixup (Zhang et al., 2019) only uses a multiplier of 1 after $\theta^{(i,0)} = 0$, thereby obtaining non-zero gradients. However, in a realistic implementation of neural networks, the multiplier of Batch Normalization can be set to 0 (Goyal et al., 2017), and down-sampling operation

can also cause 0 filled features[1][2]. Under the implementations, $\theta^{(i,0)}$ always acquires gradients with 0 values, known as the dead neuron problem, which causes failed weight updating.



(a) Weight initialized with numerical value 0. (b) Weight initialized with IDIZ_{1e-6}.

Figure 5: The last weight in a residual block of a trained ResNet. More than half of elements in (a) are not trained, which is known as the dead neuron. By contrast, IDIZ_{1e-6} successfully solves the dead neuron problem and makes all the elements in (b) trainable.

Tackling this problem, we generate small values on $\theta^{(i,0)}$ to assist in training. Recall the goal of identity-control initialization that outputs 0. Therefore, we build a calculation to get the expectation and variance of outputs approaching 0. Considering two i.i.d variables, v_1 and v_2, whose variances are $\sigma^2(v_1) = \sigma^2(v_2) = \varphi$ and means are $\mu(v_1) = \mu(v_2) = \gamma$, the variable $v = \varepsilon(v1 - v2)$ have

$$\begin{cases} \mu(v) = 0, \\ \sigma^2(v) = 2\varphi\varepsilon^2, \end{cases} \tag{12}$$

where ε is a coefficient, and $\sigma^2(v)$ will be limited to 0 when ε is sufficiently small. Assuming elements of $x^{(i)}$ are i.i.d to each other, by applying subtraction on any two elements, the result has a mean of 0, and a variance related to ε. We also take $\theta^{(i,0)} \in \mathbb{R}^{D^{\mathbf{0}}_{i+1} \times D^{\mathbf{0}}_i}$ as an instance. At first, we initialize $\theta^{(i,0)}$ with IDI_ε. Then consider two cases: (i) if $D^{\mathbf{0}}_{i+1} < D^{\mathbf{0}}_i$, setting $\theta^{(i)}_{:,D^{\mathbf{0}}_{i+1}+1:D^{\mathbf{0}}_i}$ with $\text{IDI}_{-\varepsilon}$; (ii) if $D^{\mathbf{0}}_{i+1} \geq D^{\mathbf{0}}_i$, set $\theta^{(i)}_{m,j} = -\varepsilon$, when $m\%D^{\mathbf{0}}_i = j - 1$. Therefore, we can obtain a variance of 0 by setting ε to a small value. This method is termed as IDIZ_ε, and we illustrate some cases in Figure 3. In this paper, we set $\varepsilon = 1e - 6$ everywhere. As shown in Figure 5, IDIZ_{1e-6} successfully initializes the last weight in a residual block. In addition, we also transform IDIZ_ε to a convolution form IDIZC_ε through the patch-maintain scheme.

The IDInit framework is characterized as follows: (1) **For Non-Residual Networks:** It involves directly applying $\text{IDI}\,\tau$ to fully-connected layers and $\text{IDIC}\,\tau$ to convolutional layers. (2) **For Residual Networks:** This includes two steps: (i) Implementing $\text{IDI}\,\tau$ and $\text{IDIC}\,\tau$ across all fully-connected and convolutional layers, respectively; (ii) Utilizing $\text{IDIZ}\,\varepsilon$ and $\text{IDIZC}\,\varepsilon$ for fully-connected and convolutional layers positioned at the end of residual blocks, and for the final classification layer.

4 EXPERIMENTS

In this section, we first analyze hyperparameters in Sec. 4.1. Then, we implement an ablation experiment in Sec. 4.2 to show the effect of the proposed two modifications in Sec. 3. We conduct experiments on residual convolution in Sec. 4.3. And we conduct image classification on ImageNet in Sec. 4.4. Later we conduct a text classification experiment in Sec. 4.5. At last, we employ a pre-training experiment on the large-scale dataset in Sec. 4.6 separately. We conduct experiments on non-residual convolution in Sec. C.2. We also analyze the variance amplification in Sec. C.3, weight distribution in Sec. C.4, and dynamical isometry in Sec. C.5.

4.1 EXPERIMENT FOR HYPERPARAMETERS

In this experiment, we compare IDInit with Kaiming (He et al., 2015) by analyzing the training hyperparameters, i.e., the weight decay and the learning rate. We use Cifar10. The backbone is ResNet-32, we use SGD with a momentum of 0.9. The batch size is 1024. We train models for 200 epochs. The learning rate is reduced with a cosine function. Each setting is trained 3 times to calculate the standard deviation. More details and results are in Sec. B.1 of the appendix.

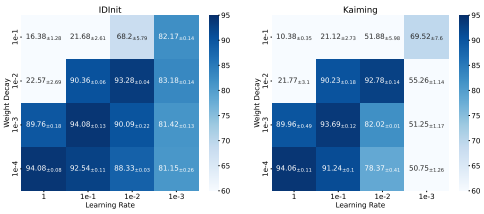
[1] https://github.com/hongyi-zhang/Fixup/blob/master/cifar/models/resnet_cifar.py

[2] https://github.com/akamaster/pytorch_resnet_cifar10/edit/master/resnet.py

Table 1: Results on Cifar10. ZerO performs worse for zero down-sampling as mentioned in Sec. 3.2. IDInit consistently facilitates rapid convergence when employed with SGD and Adam.

Initialization	56 Layer (SGD/Adam)		110 Layer (SGD/Adam)	
	Acc.	Epochs to 80% Acc.	Acc.	Epochs to 80% Acc.
Zero γ	$92.32_{\pm 0.19}$ / $87.37_{\pm 0.43}$	$57_{\pm 7}$ / $63_{\pm 4}$	$93.07_{\pm 0.28}$ / $88.30_{\pm 0.31}$	$36_{\pm 2}$ / $56_{\pm 7}$
ZerO	$90.57_{\pm 0.31}$ / $83.53_{\pm 0.42}$	$57_{\pm 3}$ / $85_{\pm 4}$	$91.71_{\pm 0.21}$ / $84.24_{\pm 0.10}$	$55_{\pm 3}$ / $76_{\pm 2}$
Fixup	$93.24_{\pm 0.82}$ / $\underline{89.50}_{\pm 0.18}$	$\underline{31}_{\pm 3}$ / $55_{\pm 3}$	$93.32_{\pm 0.23}$ / $\mathbf{90.67}_{\pm 0.12}$	$33_{\pm 3}$ / $49_{\pm 2}$
SkipInit	$92.29_{\pm 0.30}$ / $85.45_{\pm 0.74}$	$\mathbf{26}_{\pm 1}$ / $81_{\pm 3}$	$92.67_{\pm 0.16}$ / $87.18_{\pm 0.94}$	$\underline{31}_{\pm 5}$ / $70_{\pm 7}$
ReZero	$93.06_{\pm 0.54}$ / $89.26_{\pm 0.30}$	$33_{\pm 2}$ / $\underline{44}_{\pm 3}$	$94.03_{\pm 0.26}$ / $90.25_{\pm 0.20}$	$35_{\pm 5}$ / $\underline{38}_{\pm 3}$
Kaiming	$\underline{93.36}_{\pm 0.14}$ / $87.55_{\pm 0.32}$	$34_{\pm 3}$ / $50_{\pm 2}$	$\mathbf{94.06}_{\pm 0.18}$ / $87.89_{\pm 0.41}$	$33_{\pm 4}$ / $56_{\pm 3}$
IDInit	$\mathbf{93.41}_{\pm 0.10}$ / $\mathbf{90.01}_{\pm 0.32}$	$\mathbf{26}_{\pm 1}$ / $\mathbf{34}_{\pm 1}$	$\underline{94.04}_{\pm 0.24}$ / $\underline{90.53}_{\pm 0.10}$	$\mathbf{27}_{\pm 1}$ / $\mathbf{36}_{\pm 2}$

As shown in Figure 6, IDInit achieves a peak accuracy of 94.08% with a weight decay of 1e-3 and a learning rate of 1e-1. In comparison to Kaiming, IDInit demonstrates superior stability, maintaining high accuracy even when the learning rate is reduced below 1e-1. Overall, IDInit consistently delivers robust performance while maintaining stability, making it a promising candidate for practical applications.



(a) IDInit (b) Kaiming

Figure 6: The hyperparameter experiment on Cifar10. IDInit demonstrates superior adaptability across a broader range of training configurations compared to Kaiming initialization, exhibiting notable stability.

4.2 Ablation Experiment

We conduct this experiment to validate the effect of the proposed two improvements. The dataset is Cifar10 and the backbone is ResNet-20 (He et al., 2016). We run four times following settings: (i) IDInit w/o IDIC_τ and w/o IDIZC_ε; (ii) IDInit w/o IDIC_τ and w/ IDIZC_ε; (iii) IDInit w/ IDIC_τ and w/o IDIZC_ε; (iv) IDInit. For model training for 200 epochs, we employ SGD with a momentum of 0.9, a weight decay of 5e-5, and an initial learning rate of 0.1, which is adjusted using a cosine annealing schedule. Additional details and results, including the Loose condition, can be found in Sec. B.3 of the appendix.

The results are shown in Table 2. By applying the identity matrix directly, (i) obtains the lowest accuracy of 87.01% among all cases. Regarding results of (ii) and (iii), both the two settings can make significant improvements of nearly 5.89% and 3.42% from (i), respectively. And IDIZC_ε can make a deeper effect than IDIC_τ. Equipping IDIC_τ and IDIZC_ε, IDInit will improve performance further, which demonstrates our modification is efficient.

Table 2: Results of the ablation experiment on ResNet-20.

Setting	(i)	(ii)	(iii)	(iv)
Accracy	$87.01_{\pm 0.29}$	$92.9_{\pm 0.18}$	$90.43_{\pm 0.14}$	$\mathbf{93.22}_{\pm 0.05}$

4.3 Image Classification on Cifar10

In this experiment, we validate IDInit with the comparison with existing initialization, including (1) Fixup (Zhang et al., 2019); (2) SkipInit (De & Smith, 2020); (3) ReZero (Bachlechner et al., 2021); (4) Kaiming (He et al., 2015); (5) Zero γ (Setting the scale in Batch Normalization (BN) to 0) (Goyal et al., 2017); (6) ZerO. We use ResNet-56/110 as backbones on Cifar10. For analyzing convergence, we adopt both SGD and Adam optimizer for updating models. We set SGD, with the momentum 0.9, the weight decay 5e-4, and the learning rate 0.2. For Adam, the learning rate is 0.001, β_1 is 0.9 and β_2 is 0.999. The training epoch is 200.

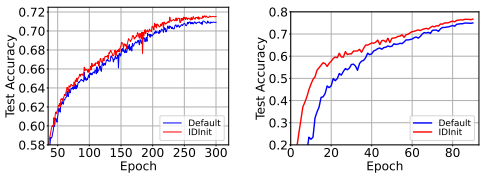
Results are shown in Table 1. Although ZerO uses the Hadamard matrix to break the rank constraint problem, it can be damaged by zero down-sampling as mentioned in Sec. 3.2. Therefore, we reclaim the importance of using IDIZ_ε and IDIZC_ε for avoiding such potential damage. Compared with baselines, IDInit derives the best accuracies in most cases. In addition, IDInit can achieve the least epochs to reach 80% accuracy in all settings, which shows a good convergence ability.

Table 4: Results of text classification on SST2 and TREC-6. The subscript G denotes the embedding layer is initialized by Glove, while W indicates Word2Vec. "Default" means the default initialization of models, specifically, Kaiming for TextCNN, and Xavier for both TextRNN and Transformer. Fixup is only applicable to the Transformer, as it is specifically designed for residual networks. Std values larger than 1.0 are marked in red. More results can be found in Table 6.

Datasets	Init.	$\text{TextCNN}_{\text{G/W}}$	$\text{TextRNN}_{\text{G/W}}$	$\text{Transformer}_{\text{G/W}}$	$\text{Average}_{\text{G/W}}$
SST2	Default	$81.40_{\pm 0.66}$ / $84.56_{\pm 0.43}$	$81.69_{\pm 0.30}$ / $84.29_{\pm 0.70}$	$80.97_{\pm 1.20}$ / $83.36_{\pm 0.76}$	$81.35_{\pm 0.72}$ / $84.07_{\pm 0.63}$
	Orthogonal	$82.24_{\pm 0.44}$ / $84.37_{\pm 0.38}$	$81.86_{\pm 0.55}$ / $84.61_{\pm 0.78}$	$82.22_{\pm 0.87}$ / $83.99_{\pm 0.23}$	$82.11_{\pm 0.62}$ / $84.32_{\pm 0.46}$
	Fixup	-	-	$78.72_{\pm 0.78}$ / $81.25_{\pm 0.27}$	-
	ZerO	$82.05_{\pm 0.67}$ / $84.26_{\pm 0.39}$	$82.03_{\pm 0.41}$ / $84.80_{\pm 0.64}$	$82.28_{\pm 0.81}$ / $82.72_{\pm 0.55}$	$82.12_{\pm 0.63}$ / $83.93_{\pm 0.53}$
	IDInit	$\mathbf{82.60}_{\pm 0.24}$ / $\mathbf{85.67}_{\pm 0.41}$	$\mathbf{82.66}_{\pm 0.16}$ / $\mathbf{85.49}_{\pm 0.33}$	$\mathbf{82.48}_{\pm 0.55}$ / $\mathbf{84.51}_{\pm 0.24}$	$\mathbf{82.58}_{\pm 0.32}$ / $\mathbf{85.22}_{\pm 0.33}$
TREC-6	Default	$90.80_{\pm 0.94}$ / $92.06_{\pm 1.00}$	$86.34_{\pm 1.04}$ / $90.52_{\pm 1.54}$	$86.68_{\pm 2.68}$ / $89.20_{\pm 1.20}$	$87.94_{\pm 1.55}$ / $90.59_{\pm 1.25}$
	Orthogonal	$90.34_{\pm 0.72}$ / $92.72_{\pm 0.84}$	$85.86_{\pm 0.90}$ / $89.88_{\pm 1.54}$	$86.90_{\pm 1.51}$ / $89.26_{\pm 0.86}$	$87.70_{\pm 0.71}$ / $90.62_{\pm 0.75}$
	Fixup	-	-	$86.95_{\pm 0.35}$ / $89.35_{\pm 0.53}$	-
	ZerO	$90.89_{\pm 0.41}$ / $92.90_{\pm 0.50}$	$\mathbf{87.24}_{\pm 0.64}$ / $88.71_{\pm 0.40}$	$86.97_{\pm 0.75}$ / $89.38_{\pm 0.64}$	$88.37_{\pm 0.60}$ / $90.33_{\pm 0.51}$
	IDInit	$\mathbf{91.22}_{\pm 0.54}$ / $\mathbf{92.94}_{\pm 0.48}$	$87.04_{\pm 0.26}$ / $\mathbf{90.60}_{\pm 0.58}$	$\mathbf{87.32}_{\pm 0.78}$ / $\mathbf{90.06}_{\pm 0.60}$	$\mathbf{88.53}_{\pm 0.53}$ / $\mathbf{91.20}_{\pm 0.55}$

4.4 Image Classification on ImageNet

In this experiments, we validate ViT-B/32 (Dosovitskiy et al., 2021), ResNet-50/152 (RN-50/152) (He et al., 2016) and Se-ResNet-50 (SRN-50) (Hu et al., 2020) as backbones on ImageNet in this experiment. For ViT-B/32, the optimizer is AdamW with a learning rate 1e-3 and a weight decay 5e-2. The training epochs is 300. We use 30 epochs for warm-up. For RN-50/152 and SRN-50, we use SGD with a learning rate 1e-1 and a weight decay 1e-4 for 90-epoch training. We use 9 epochs for warm-up. For all models, the batch size is 1024, and we apply data augment including cutmix (Yun et al., 2019) with $\alpha = 1.0$, mixup (Zhang et al., 2018) with $\alpha = 0.8$, the switching probability is 0.5 and a label smoothing with 0.1. More details and results can be found in Sec. B.5.



(a) ViT-B/32 (b) RN-50

Figure 7: Results on ImageNet. "Default" means the default initialization of models.

Results are shown in Figure 7 and Table 3. On three types of networks, i.e., ViT, ResNet and Se-ResNet, and multiple depths, IDInit always achieves faster convergence and better performance than the baseline. When training RN-50 with Adamw, the convergence of IDInit is consistently fast. Compared with RN-50, our initialization shows a faster convergence speed. IDInit has an average improvement of 0.55%, which is significant to be in practice. This experiment shows the good practicability and promising probability of IDInit, which is beneficial to the artificial intelligence community.

Table 3: Results on ImageNet. The value in brackets means "Epochs to 60% Acc". On average, IDInit enhances accuracy by 0.55% compared to the baseline and expedites model convergence by 7.4 epochs.

Model	ViT-B/32	RN-50 (Adamw)	RN-50	SRN-50	RN-152	Avg (Δ)
Default	71.05 (44)	76.20 (20)	75.70 (38)	76.30 (32)	78.76 (28)	0 (0)
IDInit	71.60 (42)	76.71 (14)	76.72 (24)	76.93 (22)	79.10 (23)	0.55 (7.4)

4.5 Text Classification

We implement text classification on SST2 (Socher et al., 2013) and TREC-6 (Li & Roth, 2002) and select TextCNN (Kim, 2014), TextRNN (Lai et al., 2015) and Transformer (Vaswani et al., 2017) for comparison. For TextCNN and TextRNN, we use AdaDelta (Zeiler, 2012) optimizer with a learning rate 1.0 and adopt Adam (Kingma & Ba, 2015) for Transformer with a learning rate 1e-4. For the embedding layer, we utilize Glove (Pennington et al., 2014) and Word2Vec(Mikolov et al., 2013) to initialize the embedding weights. All models are trained up to 10 epochs for 5 times.

As shown in Table 4, all the initialization methods can work normally. Default random initialization obtains the lowest accuracy in most cases on both SST2 and TREC-6. Orthogonal initialization always derives modest results. By contrast to baselines, IDInit can achieve the highest accuracy in all conditions. In addition, IDInit always obtains the smallest std values, showing stable performance.

4.6 PRE-TRAINING ON LANGUAGE MODEL

Pre-training plays an important role in various applications. We conduct the experiment to show the fast convergence on BERT (Devlin et al., 2019). The dataset is the concatenation of English Wikipedia and Toronto Book Corpus (Zhu et al., 2015). We train the BERT-Base for 40 epochs with 768 batch size. The optimizer is AdamW with learning rate 1e-4 and weight decay 1e-2. 32 NVIDIA V100s are used.

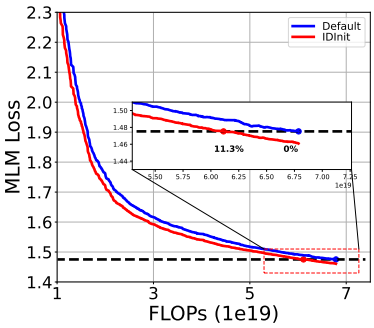


Figure 8: Results of BERT-Base.

As shown in Figure 8, "Default" means the default initialization of BERT-Base. IDInit achieves faster convergence. Specifically, IDInit shows an 11.3% acceleration ratio in terms of FLOPs. Moreover, IDInit can derive a lower loss of 1.46 in the end. As a result, IDInit is promising used in practice for enhancing convergence ability and performance.

5 DISCUSSION

The position of ReLU. Pennington et al. (2017) pointed out that non-residual networks cannot achieve dynamical isometry when using the ReLU activation function. However, in residual networks, such as $Y = W_2\text{ReLU}(W_1X) + X$, the non-linearity resides within the sub-stem of the residual block. As explored in Bartlett et al. (2019) and Hardt & Ma (2017), when the weights in the sub-stem are small, residual networks with ReLU can effectively approximate a linear network. This enables the model to follow dynamical isometry, as illustrated in Figure 18, where IDInit results in most χ values being close to 1. Moreover, Tarnowski et al. (2019) provide a theoretical perspective, suggesting that any activation function within the residual stem can support dynamical isometry. However, this behavior changes if ReLU is instead placed in the main stem, such as in $Y = \text{ReLU}(W_2W_1X + X)$. In this configuration, the network fails to maintain isometry, as noted by Pennington et al. (2017). This indicates that placing ReLU in the main stem is not advisable.

Theoretical analysis regarding the convergence rate. Theoretical exploration of the convergence rate is a critical yet challenging aspect of initialization methods. The convergence process in deep neural networks is iterative and influenced by numerous factors beyond the initialization method, including the network architecture, optimization algorithm, learning rate, batch size, and data distribution. As this area holds significant importance, further research is necessary to gain deeper insights, which will contribute to a more comprehensive understanding of initialization.

Theoretical analysis regarding the identical initalization. Figure 2 highlights the motivation behind the design of IDInit by demonstrating how identity-based initialization preserves structural bias while avoiding pitfalls such as rank constraints. Although the exact mechanism behind this improvement remains an open question, it is a promising area for future research. Investigating this mechanism further could provide valuable insights and pave the way for the development of more efficient initialization strategies, benefiting the broader research community.

6 CONCLUSION

An efficient initialization approach is crucial for training deep neural networks. In this paper, we introduce a fully identical initialization (IDInit) that is based on the identity matrix. Addressing the problems encountered when developing IDInit, i.e., dead neurons and performance degeneration, we give two concise solutions, namely using small numerical values to wipe off dead neurons and reshaping an identity-like matrix into a tensor thus increasing feature diversity, leading to a performance improvement. With good performance on wide generality, high stability, and fast convergence, IDInit is promising to be applicable in practice. In the future, we hope that this identical design can motivate the AI community to implement more novel initialization methods.

ACKNOWLEDGE

We sincerely thank everyone who contributed to improving the quality of this paper. The computations in this research were performed using the CFFF platform of Fudan University.

IMPACT STATEMENTS & LIMITATION

Impact Statements. This paper introduces IDInit, an initialization method designed to enhance stability and convergence of the training process for neural networks. This method is unlikely to have negative societal impacts.

Limitation. While IDInit demonstrates notable advancements in convergence speed and performance enhancement, it faces challenges in converging to ground truths that include negative eigenvalues. However, this drawback can be easily mitigated by incorporating momentum into the optimizer. Given that momentum is a commonly used setting, this limitation can be implicitly resolved as we show in the main context.

REFERENCES

Devansh Arpit, Víctor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets. In *NeurIPS*, pp. 10900–10909, 2019.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Gary Cottrell, and Julian J. McAuley. Rezero is all you need: fast convergence at large depth. In *UAI*, volume 161 of *Proceedings of Machine Learning Research*, pp. 1352–1361. AUAI Press, 2021.

Peter L. Bartlett, David P. Helmbold, and Philip M. Long. Gradient descent with identity initialization efficiently learns positive-definite linear transformations by deep residual networks. *Neural Comput.*, 31(3), 2019.

Yaniv Blumenfeld, Dar Gilboa, and Daniel Soudry. Beyond signal propagation: Is feature diversity necessary in deep neural network initialization? In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 960–969. PMLR, 2020.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

Soham De and Samuel L. Smith. Batch normalization biases residual blocks towards the identity function in deep networks. In *NeurIPS*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pp. 4171–4186. Association for Computational Linguistics, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

Dar Gilboa, Bo Chang, Minmin Chen, Greg Yang, Samuel S. Schoenholz, Ed H. Chi, and Jeffrey Pennington. Dynamical isometry and a mean field theory of lstms and grus. *CoRR*, abs/1901.08987, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pp. 249–256. JMLR.org, 2010.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *CVPR*, pp. 1577–1586. Computer Vision Foundation / IEEE, 2020.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. In *ICLR (Poster)*. OpenReview.net, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pp. 1026–1034. IEEE Computer Society, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778. IEEE Computer Society, 2016.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(8):2011–2023, 2020.

Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pp. 1746–1751. ACL, 2014.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, pp. 2267–2273. AAAI Press, 2015.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015.

Nannan Li, Yu Pan, Yaran Chen, Zixiang Ding, Dongbin Zhao, and Zenglin Xu. Heuristic rank selection with progressively searching tensor ring network. *Complex & Intelligent Systems*, pp. 1–15, 2021.

Xin Li and Dan Roth. Learning question classifiers. In *COLING*, 2002.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR (Workshop Poster)*, 2013.

Yu Pan, Zeyong Su, Ao Liu, Jingquan Wang, Nannan Li, and Zenglin Xu. A unified weight initialization paradigm for tensorial convolutional neural networks. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17238–17257. PMLR, 2022.

Yu Pan, Ye Yuan, Yichun Yin, Zenglin Xu, Lifeng Shang, Xin Jiang, and Qun Liu. Reusing pretrained models by multi-linear operators for efficient training. In *NeurIPS*, 2023.

Yu Pan, Ye Yuan, Yichun Yin, Jiaxin Shi, Zenglin Xu, Ming Zhang, Lifeng Shang, Xin Jiang, and Qun Liu. Preparing lessons for progressive training on language models. In *AAAI*, pp. 18860–18868. AAAI Press, 2024.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pp. 1532–1543. ACL, 2014.

Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *NIPS*, pp. 4785–4795, 2017.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *NIPS*, pp. 3360–3368, 2016.

Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7824–7835. PMLR, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pp. 1631–1642. ACL, 2013.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (Workshop)*, 2015.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1139–1147. JMLR.org, 2013.

Wojciech Tarnowski, Piotr Warchol, Stanislaw Jastrzebski, Jacek Tabor, and Maciej A. Nowak. Dynamical isometry is achieved in residual networks in a universal way for any activation function. In *AISTATS*, volume 89 of *Proceedings of Machine Learning Research*, pp. 2221–2230. PMLR, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

Maolin Wang, Yu Pan, Zenglin Xu, Xiangli Yang, Guangxi Li, and Andrzej Cichocki. Tensor networks meet neural networks: A survey and future perspectives. *arXiv preprint arXiv:2302.09019*, 2023.

Greg Yang and Samuel S. Schoenholz. Mean field residual networks: On the edge of chaos. In *NIPS*, pp. 7103–7114, 2017.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, pp. 6022–6031. IEEE, 2019.

Sergey Zagoruyko and Nikos Komodakis. Diracnets: Training very deep neural networks without skip-connections. *CoRR*, abs/1706.00388, 2017.

Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR (Poster)*. OpenReview.net, 2018.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *ICLR (Poster)*. OpenReview.net, 2019.

Jiawei Zhao, Florian Tobias Schaefer, and Anima Anandkumar. Zero initialization: Initializing neural networks with only zeros and ones. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=1AxQpKmiTc`.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, pp. 19–27. IEEE Computer Society, 2015.

A IDInit Details

A.1 Full IDInit Scheme

Here, we show the full IDInit scheme in Figure 9.

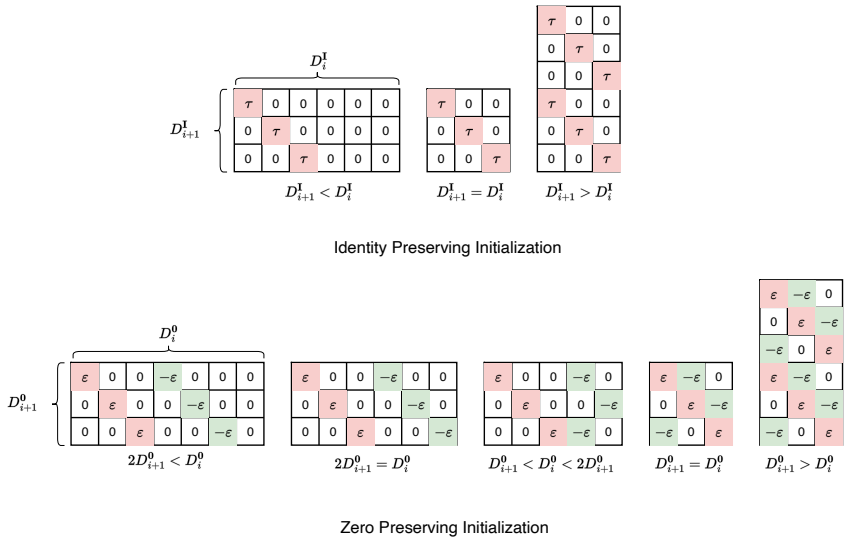


Figure 9: Illustration of IDInit with all conditions.

A.2 Analysis on Convergence

The issue of convergence was proposed by Bartlett et al. (2019). According to their study, when layers in a neural network are initialized using the identity matrix, all the weight matrix of layers will be symmetric at each step of the training process. This persistent symmetry leads to the weights of layers being the same as each other at any step, posing a significant challenge in converging to the ground truth of which eigenvalues with negative values. Our findings indicate that employing a stochastic gradient descent (SGD) approach can effectively break the symmetry which facilitates convergence, and incorporating momentum can further accelerate the convergence process. In this context, we provide formal proof demonstrating that SGD with momentum can alleviate the convergence issue.

Proof. First of all, we present a training case for a single-layer network expressed as $y = \theta x$, where $x \in \mathbb{R}^d$ represents the input, $y \in \mathbb{R}^d$ denotes the output, and $\theta \in \mathbb{R}^{d\times d}$ is the weight matrix. The weight matrix θ is initialized to the identity matrix I, denoted as $\theta^{(0)} = I$. For our loss function, we employ the Mean Squared Error (MSE) and a learning rate denoted by η. Consider two training pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$ sampled from the same dataset $\mathcal{D}$. The network is initially trained with $\{x_1, y_1\}$, and trained with $\{x_2, y_2\}$ in the next step.

In the first step, we can get the prediction as

$$\hat{y}_1 = \theta^{(0)} x_1. \tag{13}$$

The updated $\theta^{(1)}$ can be derived by

$$\begin{aligned} \Delta\theta^{(0)} &= (\hat{y}_1 - y_1)x_1^T = (\theta^{(0)}x_1 - y_1)x_1^T = (x_1 - y_1)x_1^T, \\ \theta^{(1)} &= \theta^{(0)} - \eta\Delta\theta^{(0)} = \theta^{(0)} - \eta(x_1 - y_1)x_1^T = I - \eta(x_1 - y_1)x_1^T. \end{aligned} \tag{14}$$

Therefore, in the second step, the gradient $\Delta\theta^{(1)}$ can be calculated as

$$\begin{aligned}\Delta\theta^{(1)} &= (\hat{y}_2 - y_2)x_2^T, \\ &= (\theta^{(1)}x_2 - y_2)x_2^T, \\ &= ((I - \eta(x_1 - y_1)x_1^T)x_2 - y_2)x_2^T, \\ &= x_2x_2^T - \eta x_1x_1^Tx_2x_2^T + \eta y_1x_1^Tx_2x_2^T - y_2x_2^T. \end{aligned} \tag{15}$$

While $x_2x_2^T$ is symmetric, $x_1x_1^Tx_2x_2^T$, $y_1x_1^Tx_2x_2^T$, and $y_2x_2^T$ can be asymmetric. To calculate the magnitude of the asymmetry in $\Delta\theta^{(1)}$, letting $\Omega = -\eta x_1x_1^Tx_2x_2^T + \eta y_1x_1^Tx_2x_2^T - y_2x_2^T$ denotes the asymmetric component, the magnitude of asymmetry that can be calculated as $\mathbb{E}(||\Omega - \Omega^T||_F^2)$. Assuming $x_1, x_2, y_1, y_2 \in \mathbb{R}^d$ are random vectors with entries that are i.i.d. Gaussian random variables, following $N(0, \sigma^2)$, then the magnitude of asymmetry is bounded as

$$4\eta^2d^3\sigma^8 - 4\eta^2d^2\sigma^8 + 2d^2\sigma^4 \leq \mathbb{E}(||\Omega - \Omega^T||_F^2) \leq 6\eta^2d^3\sigma^8 + 3d^2\sigma^4. \tag{16}$$

The proof can be found in Sec. A.3. As η is usually $1e-1$, and both training pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$ can be generally normalized to $\mathcal{N} \sim (0, 1)$, thereby, the symmetry of the weight can be sufficiently influenced as

$$\theta^{(2)} = \theta^{(1)} - \eta\Delta\theta^{(1)}. \tag{17}$$

When introducing a momentum $m^{(0)}$ initialized to $\Delta\theta^{(0)}$, assuming the coefficient of m is γ, $\theta^{(2)}$ will be updated as

$$\begin{aligned} m^{(1)} &= \gamma m^{(0)} + \eta\Delta\theta^{(1)}, \\ \theta^{(2)} &= \theta^{(1)} - m^{(1)} = \theta^{(1)} - \gamma m^{(0)} - \eta\Delta\theta^{(1)}. \end{aligned} \tag{18}$$

Therefore, momentum can promote the weight to become asymmetric by accumulating the asymmetry of gradients in steps and impact more when samples are increased.

As for networks of multiple layers, when their layers are asymmetric, each layer can be updated differently which breaks the convergence problem caused by the same gradients in each step (which is stated in Lemma 5 of Bartlett et al. (2019)). □

This proof primarily demonstrates that SGD with momentum can effectively resolve the issue of layers being the same in networks initialized with the identity matrix during training, which facilitates the convergence process. As illustrated in Figure 10, it is evident that layers trained using SGD are different from each other, with the momentum component amplifying the degree of this difference. By theoretically and empirically demonstrating that SGD with momentum can efficiently address this convergence problem, we hope this finding can offer valuable insights for the research community, encouraging further investigation into identity initialization and its significant role in model training.

A.3 Analysis on Asymmetry

In this section, we analyze the magnitude of asymmetry in the gradient.

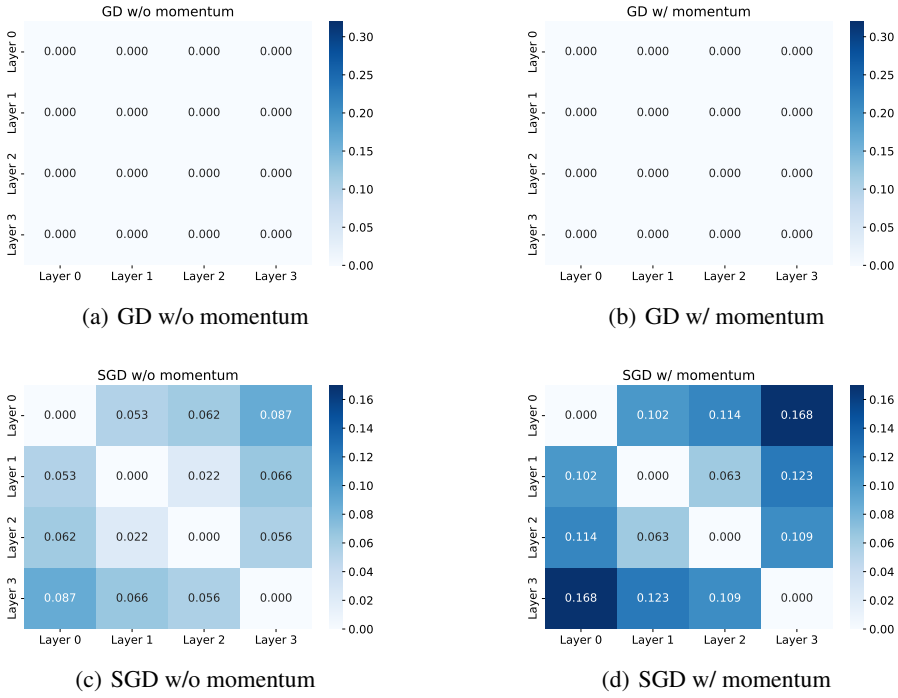
Setup and Target. Here, we assume $x_1, x_2, y_1, y_2 \in \mathbb{R}^d$ are random vectors with entries that are i.i.d. Gaussian random variables, following $N(0, \sigma^2)$. According to Eq. (4), the asymmetry in the gradient arises from:

$$\Omega = -\eta x_1x_1^Tx_2x_2^T + \eta y_1x_1^Tx_2x_2^T - y_2x_2^T. \tag{19}$$

Our target is to compute the magnitude of asymmetry that can be calculated as

$$\begin{aligned} &\mathbb{E}(||\Omega - \Omega^T||_F^2) \\ =&\mathbb{E}\{||[-\eta x_1x_1^Tx_2x_2^T + (\eta x_1x_1^Tx_2x_2^T)^T] + [\eta y_1x_1^Tx_2x_2^T - (\eta y_1x_1^Tx_2x_2^T)^T] \\ &+ [-y_2x_2^T + (y_2x_2^T)^T]||_F^2\} \end{aligned} \tag{20}$$

Lower Bound. Introducing substitutions $u = y_1 - x_1$, and $s = x_1^Tx_2 = x_2^Tx_1$, we rewrite:



(a) GD w/o momentum (b) GD w/ momentum

(c) SGD w/o momentum (d) SGD w/ momentum

Figure 10: The distance between two layers in a 4-layer network after training. In this experiment, we set the target matrix as $-I \in \mathbb{R}^{10\times 10}$. The weights are $W_0, W_1, W_2, W_3 \in \mathbb{R}^{10\times 10}$. We randomly generated 4000 data pairs $\{X_i, Y_i\}$ by $Y_i = -IX_i + \xi$, where $X_i, Y_i \in \mathbb{R}^{10}$, and ξ is noise with mean 0 and std 1e-2. We use 2000 samples for training the network. We use the other 2000 samples for testing. Batch size is 4. Mean squared error (MSE) is used as the loss function. We calculate the distance by averaging the absolute value from the difference value of two layers. Layers trained using SGD display distinct differences from one another, and the incorporation of momentum significantly increases these differences, thereby accelerating the convergence speed.

$$\mathbb{E}(||\Omega - \Omega^T||_F^2) = \mathbb{E}\{||\eta s(ux_2^T - x_2u^T) - (y_2x_2^T - x_2y_2^T)||_F^2\}, \tag{21}$$

Let $w = \eta su - y_2$, then:

$$\begin{aligned}
\mathbb{E}(||\Omega - \Omega^T||_F^2) &= \mathbb{E}\{||wx_2^T - x_2w^T||_F^2\}, && (22)\\
&= \mathbb{E}\{2(||w||^2||x_2||^2 - (w^Tx_2)^2)\}, && (23)\\
&= 2\left(\mathbb{E}[||w||^2]\mathbb{E}[||x_2||^2] - \mathbb{E}[(w^Tx_2)^2]\right), && (24)
\end{aligned}$$

Expanding and computing expectations:

$$\begin{aligned}
\mathbb{E}(||\Omega - \Omega^T||_F^2) &\geq 2((\eta^2(2d^2 - d)\sigma^6 + d\sigma^2)d\sigma^2 - \eta^2d^2\sigma^8), && (25)\\
&= 4\eta^2d^3\sigma^8 - 4\eta^2d^2\sigma^8 + 2d^2\sigma^4. && (26)
\end{aligned}$$

Upper Bound. We derive the upper bound as

$$\mathbb{E}(||\Omega - \Omega^T||_F^2) \tag{27}$$

$$=\mathbb{E}\{|||[-\eta x_1 x_1^T x_2 x_2^T + (\eta x_1 x_1^T x_2 x_2^T)^T] + [\eta y_1 x_1^T x_2 x_2^T - (\eta y_1 x_1^T x_2 x_2^T)^T] + [-y_2 x_2^T + (y_2 x_2^T)^T]||_F^2\} \tag{28}$$

According to Relaxed Triangle Inequality, there is

$$\leq 3\{\eta^2 \mathbb{E}[|| - x_1 x_1^T x_2 x_2^T + (x_1 x_1^T x_2 x_2^T)^T||_F^2] + \eta^2 \mathbb{E}[||y_1 x_1^T x_2 x_2^T - (y_1 x_1^T x_2 x_2^T)^T||_F^2] + \mathbb{E}[|| - y_2 x_2^T + (y_2 x_2^T)^T||_F^2]\} \tag{29}$$

$$\leq 3(\eta^2 d^3 \sigma^8 + \eta^2 d^3 \sigma^8 + d^2 \sigma^4) \tag{30}$$

$$=6\eta^2 d^3 \sigma^8 + 3d^2 \sigma^4 \tag{31}$$

This shows that a higher learning rate promotes greater asymmetry, further explaining the observed differences. However, a high learning rate can affect training stability. Therefore, while using a higher learning rate to reduce symmetry, it is crucial to carefully select its magnitude to maintain stability.

A.4 Proof for Theorem 3.1.

Proof. Consider a network with a single hidden layer (i.e. $L = 3$) and a batch of linearly independent samples, $x_1^{(0)} = \{x_1^{(0,1)}, \ldots, x_1^{(0,N)}\}$, with $N = D_0$. Using $\Pi_1 = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial x_1^{(3,i)}} \times x_1^{(0,i)}$, the gradients for the first update step can be written as

$$\frac{\partial \mathcal{L}}{\partial \theta^{(0)}} = \begin{pmatrix} \Pi_1 \\ \mathbf{0} \end{pmatrix} \qquad \frac{\partial \mathcal{L}}{\partial \theta^{(1)}} = \begin{pmatrix} \Pi_1 & \Pi_1 \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \qquad \frac{\partial \mathcal{L}}{\partial \theta^{(2)}} = \begin{pmatrix} \Pi_1 & \Pi_1 \end{pmatrix}. \tag{32}$$

After updating the weights with learning rate $\eta > 0$, we have

$$\theta^{(0)} = \begin{pmatrix} \boldsymbol{I} - \eta\, \Pi_1 \\ \boldsymbol{I} \end{pmatrix} \qquad \theta^{(1)} = \begin{pmatrix} \boldsymbol{I} - \eta\, \Pi_1 & -\eta\, \Pi_1 \\ \mathbf{0} & \boldsymbol{I} \end{pmatrix} \qquad \theta^{(2)} = \begin{pmatrix} \boldsymbol{I} - \eta\, \Pi_1 & -\eta\, \Pi_1 \end{pmatrix}. \tag{33}$$

As a result, the gradients of $\theta^{(1)}$ for the second update with a second batch of linearly independent samples $x_2^{(0)} = \{x_2^{(0,1)}, \ldots, x_2^{(0,N)}\}$, are given by

$$\frac{\partial \mathcal{L}}{\partial \theta^{(1)}} = \sum_{i=1}^{N} \left(\theta^{(0)} \cdot x_2^{(0,i)}\right) \times \left(\frac{\partial \mathcal{L}}{\partial x_2^{(3,i)}} \cdot \theta^{(2)\mathsf{T}}\right) \tag{34}$$

$$= \begin{pmatrix} (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2 \\ -\eta\, \Pi_1^{\mathsf{T}}\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & -\eta\, \Pi_1^{\mathsf{T}}\, \Pi_2 \end{pmatrix}, \tag{35}$$

with $\Pi_2 = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial x_2^{(3,i)}} \times x_2^{(0,i)}$. Using the gradients of the second batch to update the parameters with the same learning rate, we obtain

$$\theta^{(1)} = \begin{pmatrix} \boldsymbol{I} - \eta\, \Pi_1 - \eta\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & -\eta\, \Pi_1 - \eta\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2 \\ \eta^2\, \Pi_1^{\mathsf{T}}\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & \boldsymbol{I} + \eta^2\, \Pi_1^{\mathsf{T}}\, \Pi_2 \end{pmatrix}.$$

Consequently, the difference of the weights after two updates to the initial value, $\boldsymbol{I}$, is given by

$$\Delta\theta^{(1)} = \begin{pmatrix} -\eta\, \Pi_1 - \eta\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & -\eta\, \Pi_1 - \eta\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}})\, \Pi_2 \\ \eta^2\, \Pi_1^{\mathsf{T}}\, \Pi_2\, (\boldsymbol{I} - \eta\, \Pi_1^{\mathsf{T}}) & \eta^2\, \Pi_1^{\mathsf{T}}\, \Pi_2 \end{pmatrix}.$$

Assuming that the gradients $\frac{\partial \mathcal{L}}{\partial x_1^{(3)}}$ and $\frac{\partial \mathcal{L}}{\partial x_2^{(3)}}$ are also linearly independent, $\mathrm{rank}(\Pi_1) = \mathrm{rank}(\Pi_2) = D_0$. Due to Sylvester's rank inequality, we can conclude that also $\mathrm{rank}(\Pi_1\, \Pi_2) = D_0$. As a result, the lower-right part of the difference has rank D_0, from which we can conclude that $\mathrm{rank}(\Delta\theta^{(1)}) \geq D_0$. □

A.5 IMPLEMENTING IDINIT ON ATTENTION LAYER IN TRANSFORMER

In this part, we show the way to initialize the attention layer with IDInit. Prior to that, formulating an attention layer as

$$\mathrm{Att}(Q, K, V) = \mathrm{softmax}(\frac{QW^QW^KK}{\sqrt{d}})VW^VW^O, \tag{36}$$

where Q is the query matrix, K means the key matrix, V denotes the value matrix, W^Q, W^K and W^V represents the weights for Q, K, and V respectively, and W^O is the output transformation. Following the instruction of IDInit in Sec. 3, we firstly use IDI_τ to initialize W^Q, W^K, W^V and W^O. And then, we use $\mathrm{IDIZ}_\varepsilon$ to initialize the last fully-connected layer W^O. The τ and ε are consistently set with the paper content to 1 and 1e-6, respectively.

A.6 DETAILS OF PATCH-MAINTAIN CONVOLUTION

We illustrate the figure to show the comparison between channel-maintain convolution and patch-maintain convolution in Figure 11.

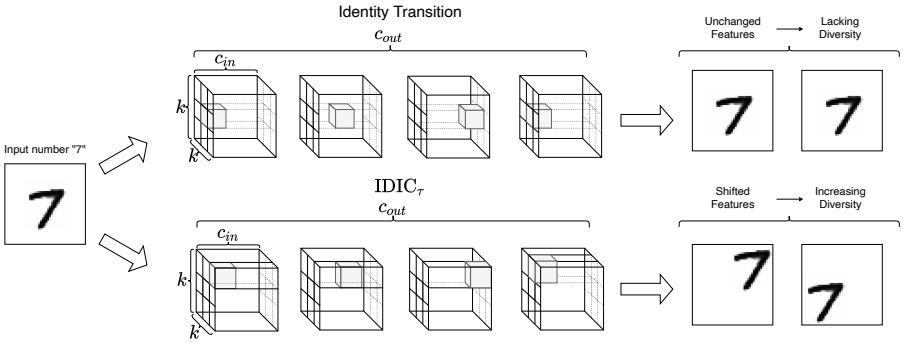


Figure 11: A case of number "7" on Identical Convolution Layer. The upper sub-figure maintains the identity transition. The under sub-figure is IDIC_τ initialization that shifts features for increasing diversity. More feature diversity from IDIC_τ is beneficial for improving model performance.

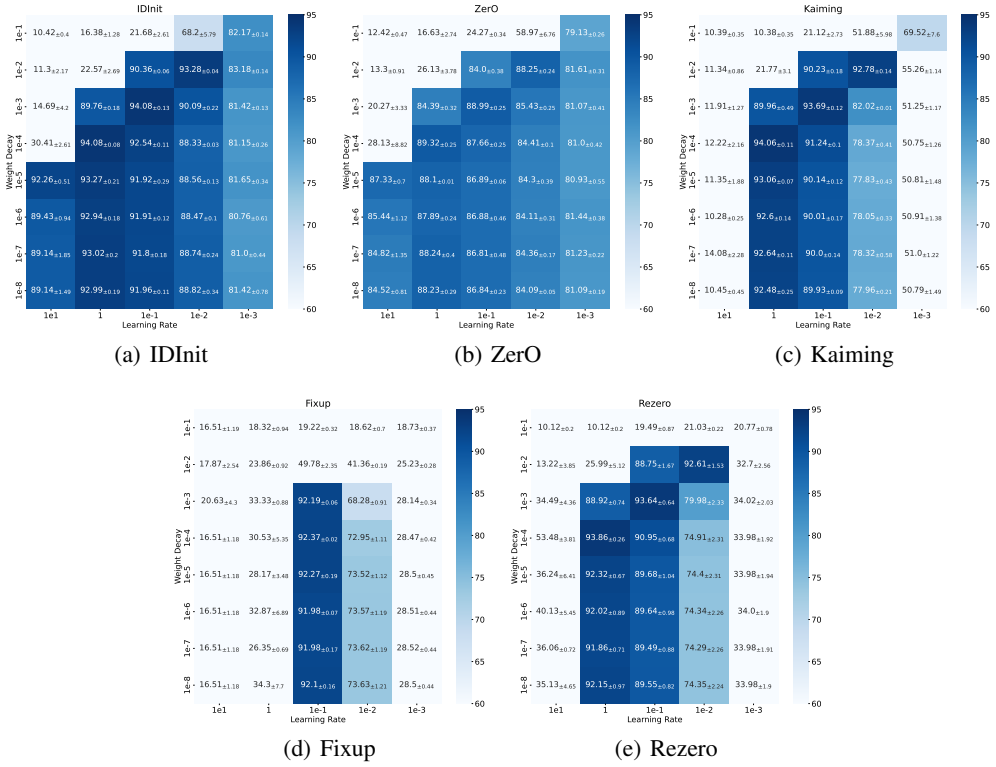
B DETAILED SETTINGS OF EXPERIMENTS

In this paper, for ReLU activated networks, τ is set to $\sqrt{2}$ for the first layer in a network and 1 for other IDI_τ / IDIC_τ initializing layers, while for tanh-activated networks, all IDI_τ is set to 1, and ε is $1e-6$ for all $\mathrm{IDIZ}_\varepsilon$ / $\mathrm{IDIZC}_\varepsilon$ initializing layers.

B.1 EXPERIMENT FOR HYPERPARAMETERS

In this experiment, we compare IDInit with other initialization methods, including (1) Fixup; (2) ReZero; (3) Kaiming; and (4) Zero, by analyzing the training hyperparameters, i.e., the weight decay and the learning rate. We use Cifar10. The backbone is ResNet-32, we use SGD with a momentum of 0.9. The batch size is 1024. We train models for 200 epochs. The learning rate is reduced with a cosine function. Each setting is trained 3 times to calculate the standard deviation.

We scanned the learning rate from 1e-3 to 1e1 and weight decay from 1e-8 to 1e-1, ensuring that the best-performing hyperparameters are not at the corners or edges of the grid. As shown in Figure 12, IDInit achieves a peak accuracy of 94.08% with a weight decay of 1e-3 and a learning rate of 1e-1. In comparison to other initialization methods including Kaiming, Fixup, and Rezero, IDInit demonstrates superior stability, maintaining high accuracy even when the learning rate is reduced below 1e-1. Although ZerO exhibits comparable stability at lower learning rates owing to its Hadamard matrix's ability to sustain dynamics, it underperforms at higher learning rates due to the dead neurons caused by the zero weights in its residual stems. Fixup, on the other hand, lacks stability by



(a) IDInit (b) ZerO (c) Kaiming

(d) Fixup (e) Rezero

Figure 12: The expanded hyperparameter experiment on Cifar10 on ResNet-32.

eliminating batch normalization, rendering it unsuitable for high learning rates. Overall, IDInit consistently delivers robust performance while maintaining stability, making it a promising candidate for practical applications.

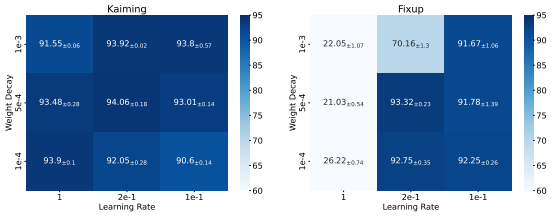
B.2 Details of Image Classification on Cifar10 Experiment

In this experiment, we validate the proposed initialization with the comparison with existing initialization, including (1) Fixup; (2) SkipInit; (3) ReZero; (4) Kaiming; (5) Zero γ (Setting the scale in Batch Normalization (BN) to 0). We use ResNet-56/110 as backbones on Cifar10. For analyzing convergence, we adopt both SGD and Adam optimizer for updating models. We set SGD, with the momentum 0.9, the weight decay 5e-4, and the learning rate 0.2. For Adam, the learning rate is 0.001, β_1 is 0.9 and β_2 is 0.999. We train models for 200 epochs. The learning rate is reduced with a cosine function. The experiment is conducted on one Nvidia A100.

We perform a detailed hyperparameter analysis for ResNet-110, evaluating the learning rates $\{$1, 2e-1, 1e-1$\}$ and weight decays $\{$1e-4, 5e-4, 1e-3$\}$ on the standard baseline Kaiming and the more fragile Fixup method. As shown in Figure 13, both Kaiming and Fixup achieve optimal accuracy with a learning rate of 2e-1 and a weight decay of 5e-4. However, Fixup fails to train with a learning rate of 1. Consequently, selecting a learning rate of 2e-1 and a weight decay of 5e-4 as the training hyperparameters in Section 4.3 is justified.

B.3 Details of Ablation Experiment

The dataset is Cifar10 and the backbone is ResNet-20. We choose SGD with momentum 0.9, weight decay 5e-4, and learning rate 0.1 to train the models for 200 epochs. The learning rate is reduced with a cosine function. And data-augment mixup is applied. The experiment is conducted on one Nvidia A100.



(a) Kaiming (b) Fixup

Figure 13: The tuning hyperparameters on Cifar10 on ResNet-110.

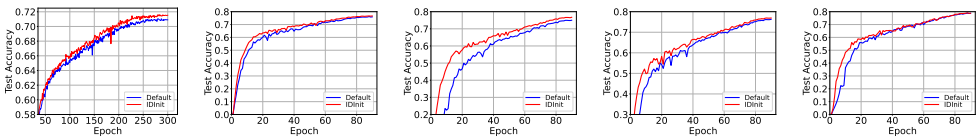
The extended analysis in Table 5 shows that the Loose condition, along with the components IDIC and IDIZ, contributes independently to performance improvements. Furthermore, the combination of these components yields the most significant results. Across all comparison pairs—specifically, settings 1/4, 2/6, 3/7, and 5/8—the Loose condition consistently demonstrates performance improvements. This highlights its practical value and its role in enhancing the overall effectiveness of the initialization methods.

Table 5: Analysis of components.

Component	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6	Setting 7	Setting 8
Loose				✓		✓	✓	✓
IDIC			✓		✓		✓	✓
IDIZ		✓			✓	✓		✓
Accuracy	$86.12_{\pm 0.52}$	$92.68_{\pm 0.08}$	$89.47_{\pm 0.24}$	$87.01_{\pm 0.29}$	$92.95_{\pm 0.21}$	$92.9_{\pm 0.18}$	$90.43_{\pm 0.14}$	$\mathbf{93.22}_{\pm 0.05}$

B.4 Details of Text Classification Experiment

We also explore performance networks on text classification datasets including SST2, SST5 (Socher et al., 2013) and TREC-6, and we select TextCNN (Kim, 2014), TextRNN (Lai et al., 2015) and Transformer (Vaswani et al., 2017) for comparison. For TextCNN and TextRNN, we use AdaDelta (Zeiler, 2012) optimizer with a learning rate 1.0 and adopt Adam (Kingma & Ba, 2015) for Transformer with a learning rate 1e-4. For the embedding layer, we utilize Glove (Pennington et al., 2014) and Word2Vec (Mikolov et al., 2013) to initialize the embedding weights. All models are trained up to 10 epochs, and we run all the random initialization 5 times. The experiment is conducted on one Nvidia A100.



(a) ViT-B/32 (b) RN-50 (Adamw) (c) RN-50 (d) SRN-50 (e) RN-152

Figure 14: Results on ImageNet. "Default" means the default initialization of models. RN-50 (Adamw) means that ResNet-50 is trained with the same optimizer Adamw as the ViT-B/32.

B.5 Details of Image Classification on ImageNet Experiment

In this experiment, we use ImageNet for validation. We use ViT-B/32 (Dosovitskiy et al., 2021), ResNet-50/152 (RN-50/152) and Se-ResNet-50 (SRN-50) as backbones. For ViT-B/32 that inputs 32×32 patch window, the optimizer is AdamW with a learning rate 1e-3 and a weight decay of 5e-2. And the batch size is 1024. The epoch for training is 300. We use 30 epochs for warm-up. The input

Table 6: Results of text classification on SST2 and TREC-6. The subscript G denotes the embedding layer is initialized by Glove, while W indicates Word2Vec. "Default" means the default initialization of models, specifically, Kaiming for TextCNN, and Xavier for both TextRNN and Transformer. Fixup, ReZero and SkipInit are only applicable to the Transformer, as it is specifically designed for residual networks. Std values larger than 1.0 are marked in red.

Datasets	Init.	$\text{TextCNN}_{\text{G/W}}$	$\text{TextRNN}_{\text{G/W}}$	$\text{Transformer}_{\text{G/W}}$	$\text{Average}_{\text{G/W}}$
SST2	Default	$81.40_{\pm 0.66}$ / $84.56_{\pm 0.43}$	$81.69_{\pm 0.30}$ / $84.29_{\pm 0.70}$	$80.97_{\pm 1.20}$ / $83.36_{\pm 0.76}$	$81.35_{\pm 0.72}$ / $84.07_{\pm 0.63}$
	Orthogonal	$82.24_{\pm 0.44}$ / $84.37_{\pm 0.38}$	$81.86_{\pm 0.55}$ / $84.61_{\pm 0.78}$	$82.22_{\pm 0.87}$ / $83.99_{\pm 0.23}$	$82.11_{\pm 0.62}$ / $84.32_{\pm 0.46}$
	Fixup	-	-	$78.72_{\pm 0.78}$ / $81.25_{\pm 0.27}$	-
	ReZero	-	-	$81.67_{\pm 0.77}$ / $82.32_{\pm 0.51}$	-
	SkipInit	-	-	$82.30_{\pm 0.47}$ / $84.12_{\pm 0.75}$	-
	ZerO	$82.05_{\pm 0.67}$ / $84.26_{\pm 0.39}$	$82.03_{\pm 0.41}$ / $84.80_{\pm 0.64}$	$82.28_{\pm 0.81}$ / $82.72_{\pm 0.55}$	$82.12_{\pm 0.63}$ / $83.93_{\pm 0.53}$
	IDInit	$\mathbf{82.60}_{\pm 0.24}$ / $\mathbf{85.67}_{\pm 0.41}$	$\mathbf{82.66}_{\pm 0.16}$ / $\mathbf{85.49}_{\pm 0.33}$	$\mathbf{82.48}_{\pm 0.55}$ / $\mathbf{84.51}_{\pm 0.24}$	$\mathbf{82.58}_{\pm 0.32}$ / $\mathbf{85.22}_{\pm 0.33}$
TREC-6	Default	$90.80_{\pm 0.94}$ / $92.06_{\pm 1.00}$	$86.34_{\pm 1.04}$ / $90.52_{\pm 1.54}$	$86.68_{\pm 2.68}$ / $89.20_{\pm 1.20}$	$87.94_{\pm 1.55}$ / $90.59_{\pm 1.25}$
	Orthogonal	$90.34_{\pm 0.72}$ / $92.72_{\pm 0.84}$	$85.86_{\pm 0.90}$ / $89.88_{\pm 1.54}$	$86.90_{\pm 1.51}$ / $89.26_{\pm 0.86}$	$87.70_{\pm 0.71}$ / $90.62_{\pm 0.75}$
	Fixup	-	-	$86.95_{\pm 0.35}$ / $89.35_{\pm 0.53}$	-
	ReZero	-	-	$86.92_{\pm 0.98}$ / $89.36_{\pm 0.52}$	-
	SkipInit	-	-	$83.59_{\pm 0.61}$ / $87.10_{\pm 0.41}$	-
	ZerO	$90.89_{\pm 0.41}$ / $92.90_{\pm 0.50}$	$\mathbf{87.24}_{\pm 0.64}$ / $88.71_{\pm 0.40}$	$86.97_{\pm 0.75}$ / $89.38_{\pm 0.64}$	$88.37_{\pm 0.60}$ / $90.33_{\pm 0.51}$
	IDInit	$\mathbf{91.22}_{\pm 0.54}$ / $\mathbf{92.94}_{\pm 0.48}$	$87.04_{\pm 0.26}$ / $\mathbf{90.60}_{\pm 0.58}$	$\mathbf{87.32}_{\pm 0.78}$ / $\mathbf{90.06}_{\pm 0.60}$	$\mathbf{88.53}_{\pm 0.53}$ / $\mathbf{91.20}_{\pm 0.55}$

image size is 224×224. The dropout rates of the embedding layer and the network layer are all 0.1. For RN-50/152 and SRN-50, the optimizer is SGD with a learning rate 1e-1 and a weight decay of 1e-4. And the batch size is 1024. The epoch for training is 90. We use 9 epochs for warm-up. The input image size is 160×160 for the front 35 epochs and 224×224 for the remaining epochs. For all models, we apply data-augment including cutmix (Yun et al., 2019) with $\alpha = 1.0$, mixup (Zhang et al., 2018) with $\alpha = 0.8$, the switching probability is 0.5 and a label smoothing with 0.1. The experiment is conducted on 4 Nvidia A100.

To further compare with other identity-control methods, we conducted experiments on ResNet-50 using Fixup and Zero. As shown in Table 7, IDInit outperforms both Fixup and ZerO, demonstrating its superior performance on large-scale datasets.

Table 7: Comparison among Default, Fixup, ZerO and IDInit initialized ResNet-50 on ImageNet.

Init.	Epochs to 60% Accuracy	Accuracy
Default	38	75.70
Fixup	24	75.83
ZerO	30	75.64
IDInit	24	76.72

B.6 Details of Pre-Training on Language Model

Pre-training plays an important role in various applications. We conduct the experiment to show the fast convergence on BERT (Devlin et al., 2019). The dataset is the concatenation of English Wikipedia and Toronto Book Corpus Zhu et al. (2015). We train the BERT-Base for 40 epochs with 768 batch size. The optimizer is set to AdamW with learning rate 1e-4 and weight decay 1e-2. 32 NVIDIA V100s are used.

C Additional Experiments

We provide additional experiments to further validate IDInit. τ and ε are set the same as Sec. B.

C.1 Validation on the Linear Structure

This experiment is conducted on MNIST. We use five linear layers named Liner-5 whose hidden layers are all of dimension 512. The optimizer is SGD with momentum 0.9, weight decay 5e-4,

Table 8: Results of Linear-5 on MNIST. "Default" means the default initialization of models where Xavier is for Linear-5-tanh and Kaiming is adopted for Linear-5-ReLU.

Init.	Linear-5-tanh	Linear-5-ReLU
Default	98.26	98.21
IDInit	**98.32**	**98.4**

and a learning rate 1e-1. The learning rate scheduler adopts a cosine reduction strategy. We run the model in 30 epochs on one Nvidia A100. We both consider Linear-5-tanh and Linear-5-ReLU which consist of Linear-5, and tanh and ReLU activation functions, respectively. The experiment is conducted on one Nvidia A100.

As shown in Table 8, IDInit can achieve the highest accuracy in both different tanh and ReLU conditions. The results show the ability of our proposed method to train a model with only fully-connected layers.

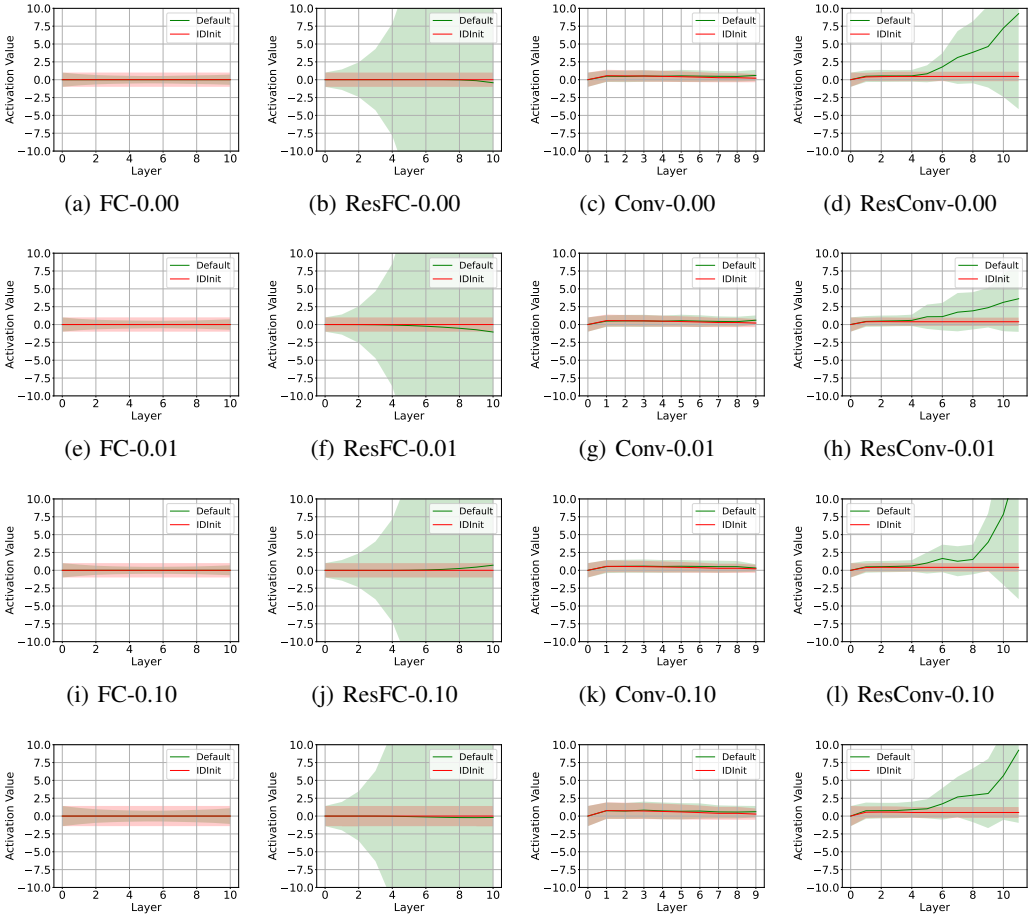
C.2 Validation on Non-Residual Convolution

We use this experiment to show IDInit can achieve a good initial state for training on non-residual convolutional networks. In this experiment, we use AllConv (Springenberg et al., 2015) which consists of nine convolutional layers as the backbone network. We show the structure of AllConv in Table 9. The dataset is Cifar10. The optimizer is Stochastic Gradient Descent (SGD) with momentum 0.9, weight decay 5e-4, and learning rate 1e-1. The learning rate scheduler adopts a warm-up cosine reduction strategy. We run the model in 300 epochs on one Nvidia A100. We adopt Kaiming initialization and IDInit w/o IDIC_τ initialization for comparison. Since there is no residual connection, we do not consider the IDIZC_ε function in this experiment. For each initialization, we have run them with 0, 10, 20, 30, 40, 50, and 60 warm-up epochs. The experiment is conducted on one Nvidia A100.

Table 9: Architectures of the tensorial All-Conv networks. Window means the convolutional kernel window size. Channels indicate $\mathbf{c}_{in}$ and $\mathbf{c}_{out}$ of a standard convolutional kernel $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{\mathbf{c}_{in} \times \mathbf{c}_{out} \times k \times k}$. The avg pool denotes the average pooling operation.

Layer	Window	Channels
conv1	3×3	3× 96
conv2	3×3	96× 96
conv3	3×3	96× 96
conv4	3×3	96× 192
conv5	3×3	192× 192
conv6	3×3	192× 192
conv7	3×3	192× 192
conv8	1×1	192× 192
conv9	1×1	192× 10 avg pool

Results are shown in Figure 16, without a warm-up strategy which is a strong trick for training, both Kaiming and IDInit w/o IDIC_ε fail to train the model. By contrast, our initialization can train AllConv and maintain the highest performance in all situations, showing a strong effect on stability and performance. As IDInit w/o IDIC_ε performs poorly, we demonstrate the patch-maintain strategy



(a) FC-0.00 (b) ResFC-0.00 (c) Conv-0.00 (d) ResConv-0.00

(e) FC-0.01 (f) ResFC-0.01 (g) Conv-0.01 (h) ResConv-0.01

(i) FC-0.10 (j) ResFC-0.10 (k) Conv-0.10 (l) ResConv-0.10

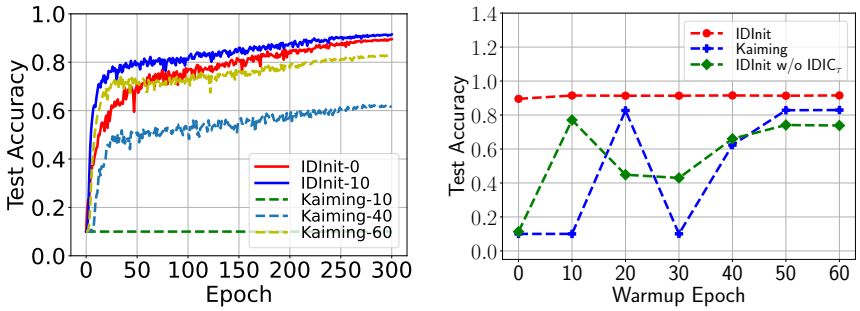
(m) FC-1.00 (n) ResFC-1.00 (o) Conv-1.00 (p) ResConv-1.00

Figure 15: Results of the analysis on variance propagation. The numerical value after the model name means the standard derivation of the noise. "Default" means the default initialization of models, specifically, Xavier for FC and ResFC, and Kaiming for Conv and ResConv. The default methods can only work on non-residual networks FC and Conv, however, fail on residual networks ResFc and ResConv, for cause instability with giant standard derivation. By contrast, IDInit can consistently transit data-flow in an appropriate scale on all models and various noises, which shows sufficient robustness, and can provide models with stable and efficient training.

mentioned in Sec. 3.1.3 can be good for increasing feature diversity. This experiment shows the identical method can be a feasible initialization for non-residual networks.

C.3 Analysis on Variance Propagation

Here we conduct an experiment on Cifar10 to demonstrate data-flow will keep stable. We use 4 types of networks: (1) FC: 10-layer fully-connected layers; (2) ResFC: 10 residual blocks (two fully-connected layers in a block); (3) Conv: 9-layer AllConv in Sec. C.2; (4) ResConv: 10 residual blocks (two convolutional layers in a block). For (1) and (2) two fully-connected networks, we reshape Cifar10 data as $\mathbf{X} \in \mathbb{R}^{32\times 96}$ as input and does not use any activation function. For (1), hidden lengths are $\{200, 400, 600, 800, 1000, 1000, 800, 600, 400, 200\}$. For (2), hidden lengths are all set to 96. For (3) and (4) two convolution networks, we directly input images to them, and use ReLU as the activation function. For (3), we directly use AllConv as shown in Table 9. For (4), we first use convolution to transfer an image to 16 channels, and then set the channels of all convolution within residual blocks to 16. For comparison, we use Xavier for (1) and (2), and Kaiming for (3) and (4) in



(a) Test Top-1 (b) Test Best

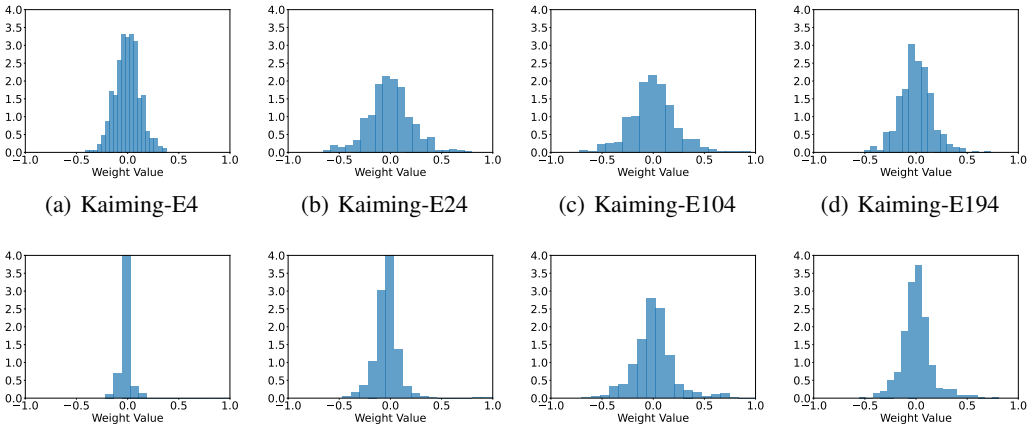
Figure 16: Results of AllConv on Cifar10. The number behind the initialization denotes the warm-up epochs.

terms of the activation function. We also employ noises with 0 mean, and $\{0.00, 0.01, 0.10, 1.00\}$ for comparing robustness. In the experiment, we run 500 rounds for each model. The experiment is conducted on one Nvidia A100.

Results are shown in Figure 15. The regular methods Xavier and Kaiming can only work on non-residual networks. On residual networks, they both cause giant standard derivation, leading to instability. By contrast, the proposed IDInit can consistently transit data-flow in an appropriate scale on all models and various noises, which shows sufficient robustness, and can provide models with stable and efficient training.

C.4 Analysis on Weight Distribution

In this experiment, we conduct an experiment on Cifar10 with ResNet-20 to show the weight distribution of IDInit. We use an SGD optimizer with a learning rate 0.2, and weight decay 5e-4. The batch size is 1024. Training epochs are 200. The learning rate is reduced with a cosine function. The experiment is conducted on one Nvidia A100.



(a) Kaiming-E4 (b) Kaiming-E24 (c) Kaiming-E104 (d) Kaiming-E194

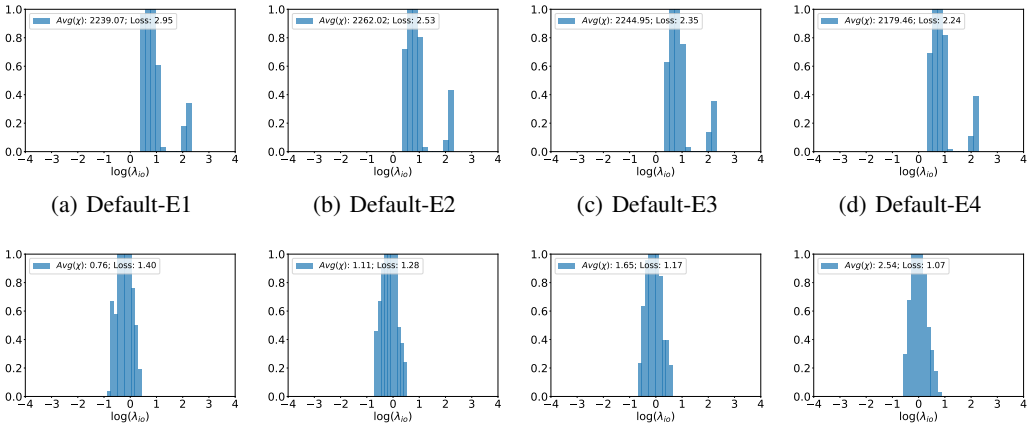
(e) IDInit-E4 (f) IDInit-E24 (g) IDInit-E104 (h) IDInit-E194

Figure 17: Histograms of the first convolution weights in ResNet-20. "E" means the epoch index. IDInit contains more zero values in each epoch compared with Kaiming initialization.

The results are shown in Figure 17, weights initialized with IDInit are almost full of zero at the beginning, while Kaiming uses a Gaussian distribution. At the end of the training, IDInit still contains more zero values than Kaiming, which is beneficial for memory occupation since a 0 value will not cost memory space.

C.5 ANALYSIS ON INPUT-OUTPUT JACOBIAN

Here we conduct an experiment on Cifar10 with 64 blocks in Figure 1 to demonstrate IDInit follows the dynamical isometry. We use the open-source code[3]. We remove batch normalization for the more clear difference between IDInit and Kaiming. We use an Adagrad optimizer with a learning rate 0.01. The batch size is 100. The activation is ReLU. The experiment is conducted on one Nvidia A100.



(a) Default-E1 (b) Default-E2 (c) Default-E3 (d) Default-E4

(e) IDInit-E1 (f) IDInit-E2 (g) IDInit-E3 (h) IDInit-E4

Figure 18: Histograms of log singular values ($\log(\lambda_{io})$) for the input-output Jacobian. "E" means the epoch index. Compared with Default initialization, IDInit has a significantly smaller squared singular value χ, which can achieve a faster reduction of the loss.

As shown in Figure 18, Default initialization cause a high squared singular value χ, reaching more than 2000. Compared to Default, IDInit only derives χ around 1, indicating correspondence to the dynamical isometry. In addition, the loss of IDInit decreases faster than Default, which shows a good convergent ability.

C.6 FAILURE OF LONG RESIDUAL STEM

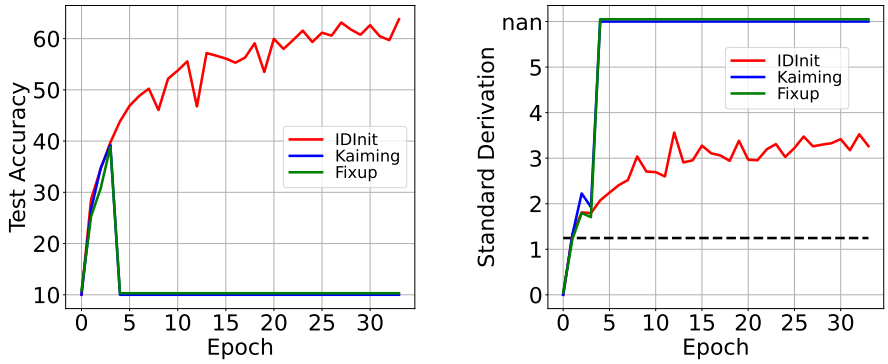
We conduct this experiment to show the failure case when the residual stem is long to show the importance of the stability of the residual stem. In this experiment, we conduct an experiment on Cifar10. We use a residual network named Res-112 as in Table 10. We set 109 layers in the residual stem. Batch normalization is not applied for fairly validating the stability of initialization methods. We use an SGD optimizer with a learning rate 0.2, and weight decay 1e-8. The batch size is 768. Training epochs are 35. The learning rate is reduced with a cosine function. One Nvidia A100 is used.

Results are shown in Figure 19. When the network is trained for 4 epochs, both Kaiming and Fixup fail to train the network, since the standard derivations of their outputs explode. By contrast, IDInit successfully trains this network and the standard derivation of the output converges to a stable value. This experiment demonstrates the ability of IDInit to stabilize the residual stem, which can benefit the training of the whole network.

C.7 EXPERIMENT ON GPT-BASE-MOE

We conducted experiments on GPT-Base-MOE, modifying the GPT-Base with 8 experts. The training settings mainly follow Pan et al. (2023). The results, shown in Figure 20, indicate that IDInit can achieve 20% faster performance compared to the default random initialization, demonstrating the superior performance of IDInit.

[3] `https://github.com/tbachlechner/ReZero-examples/blob/master/ReZero-Deep_Fast_NeuralNetwork.ipynb`



(a) Test Accuracy (b) Standard Derivation

Figure 19: Result of the experiment on the residual network with the long residual stem. Figure 19(a) shows the accuracy of different initialization. Figure 19(b) shows the standard derivations of the outputs of networks with different initialization methods. The black dash line is the standard derivation of the network input.

Table 10: Architectures of Res-112. Window means the convolutional kernel window size. Channels indicate $\mathbf{c}_{in}$ and $\mathbf{c}_{out}$ of a standard convolutional kernel $\mathcal{C} \in \mathbb{R}^{\mathbf{c}_{in} \times \mathbf{c}_{out} \times k \times k}$. The avg pool denotes the average pooling operation. Linear means a linear layer.

Layer	Window	Channels
conv1	3×3	3×16
Residual Block	3×3	[16×16]×18
	3×3	16×32
		[32×32]×17
	3×3	32×64
		[64×64]×17
	3×3	64×64
conv2	3×3	64×64 avg pool
Linear		64×10

C.8 Experiment on DiT

We train DiT-S/4 on ImageNet using the provided code[4]. The experiment is conducted using the default training settings. As illustrated in Figure 21, IDInit consistently achieves faster convergence compared to the default initialization.

D Dynamical Isometry in IDInit

Following Bachlechner et al. (2021), we utilize a simple example of the mechanism that dynamical isometry helps IDInit to obtain a fast convergence. Considering a L-layer network with a simple special case of Eq. (1):

$$x^{(L)} = (r + w^{(2)} w^{(1)})^L x^{(0)}, \tag{37}$$

[4] https://github.com/facebookresearch/DiT

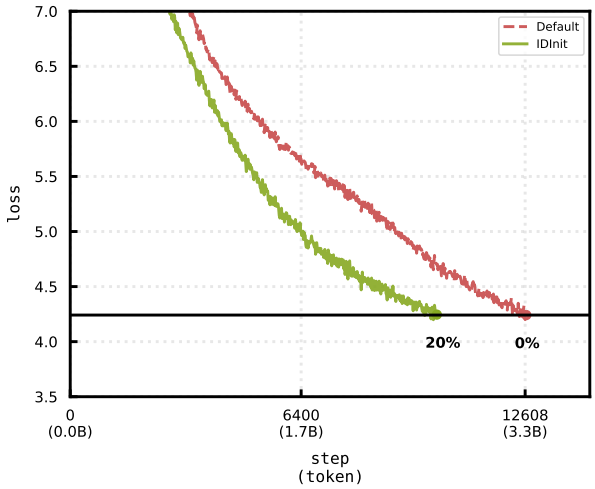


Figure 20: Pretraining on GPT-Base-MOE. IDIinit can achieve 20% after than Default initialization.

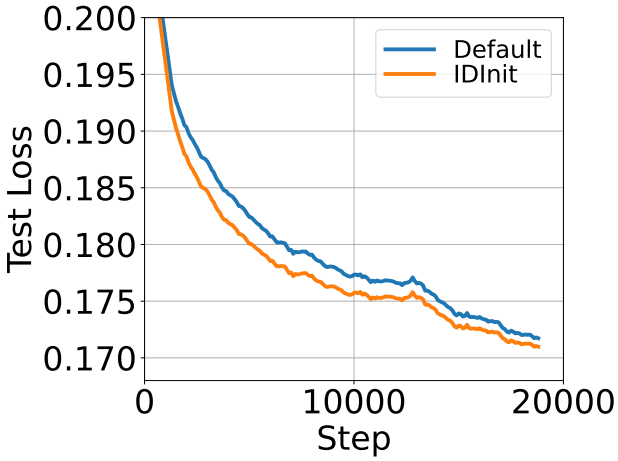


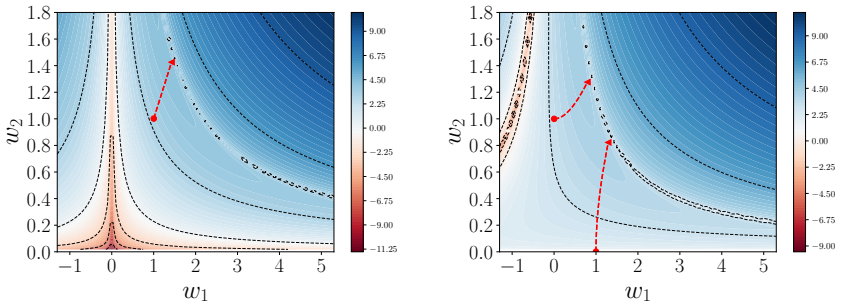
Figure 21: Training on DiT-S/4.

where $w^{(1)}$ and $w^{(2)}$ denote the first weight and last weight in a residual stem respectively, and $x^{(*)}$ is the feature in layers. $r \in \{0, 1\}$ determines residual connection. Specifically, $r = 0$ and $r = 1$ represent non-residual and residual conditions respectively. The Jacobian of Eq. (37) is $J_{0L} = (r + w^{(2)}w^{(1)})^L$. Obviously, identity transition on both non-residual and residual settings, namely $\{r = 0, w^{(2)} = w^{(1)} = 1\}$ and $\{r = 1, w^{(1)} = 1, w^{(2)} = 0\}$ respectively, will achieve $J_{0L} = 1$, which conforms to the dynamical isometry mechanism that helps improving training ability (Pennington et al., 2017). Further, we delve into a gradient update analysis. Following gradient descent, w_1 can be updated with

$$\Delta w^{(1)} = -\lambda L w^{(2)} x^{(0)} (r + w^{(2)} w^{(1)})^{L-1} \partial_x R(x)|_{x=x^{(L)}}, \tag{38}$$

where R means the loss function, and λ is a learning rate. As $w^{(1)}$ and $w^{(2)}$ are equivalent in Eq. (37), $w^{(2)}$ can be updated similar to Eq. (38). When $w^{(1)} = 1$, updates are required less than 1. Therefore, the learning rate is constrained to

$$\begin{cases} \lambda \propto L^{-1}, & \text{if non-residual,} \\ \lambda \propto L^{-1}(1 + w^{(2)})^{L-1}, & \text{if residual.} \end{cases} \tag{39}$$



(a) Non-Residual Plot. (b) Residual Plot.

Figure 22: Contour plots of the log gradient norm $\log ||\partial R||_2$ on non-residual and residual networks. $w^{(1)}$ and $w^{(2)}$ are both weights. The training process set as Bachlechner et al. (2021), which is conducted on ground-truth $x^{(L)} = 50 \times x_0$ via gradient descent using a training set of $x_0 = \{1., 1.1, ..., 1.8\}$. (a) shows $\{w^{(2)} = w^{(1)} = 1\}$ can avoid poorly conditioned regions around 0, and converge to $w^{(1)}w^{(2)} = 2.19$. (b) cares about two initial position $\{w^{(1)} = 0, w^{(2)} = 1\}$ and $\{w^{(2)} = 1, w^{(1)} = 0\}$. The two points' trajectories do not also pass the poor regions around $w^{(1)} = -1, w^{(2)} = 1$ and converge to the solution $w^{(1)}w^{(2)} = 1.19$.

For the non-residual condition, the learning rate is polynomial to L, thereby insensitive to the depth. By contrast, in the residual block, $w^{(2)} >> 0$ will cause learning rate exponentially small and $w^{(2)} = -1$ also cause gradient diffusion. On this condition, setting $w^{(2)} = 0$ can be a good solution for avoiding large output and restricting gradients in a suitable norm. Besides, it is feasible to update $w^{(2)}$ with the first non-trial step

$$w^{(2)} = -\lambda L w^{(1)} x^{(0)} \partial_x R(x)|_{x=x^{(L)}}, \tag{40}$$

and will converge with a learning rate that is polynomial in the depth L of the network. We plot the training dynamics in Figure 22, and use this simple example to illustrate the mechanism of IDInit, which is always a well-conditioned position for training.