IDENTICAL INITIALIZATION: A UNIVERSAL APPROACH TO FAST AND STABLE TRAINING OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

A well-conditioned initialization is beneficial for training deep neural networks. However, existing initialization approaches do not simultaneously show robustness and universality. Specifically, even though the widely-used Xavier and Kaiming initialization approaches can generally fit a variety of networks, they fail to train residual networks without Batch Normalization for calculating an inappropriate scale on data-flow. On the other hand, some literature design stable initialization (e.g., Fixup and ReZero) based on dynamical isometry, an efficient learning mechanism. Nonetheless, these methods are specifically designed for either a non-residual structure or a residual block only, and even include extra auxiliary components, limiting their applicable range. Intriguingly, we find that the identity matrix is a feasible and universal solution to the aforementioned problems, as it adheres to dynamical isometry while remaining applicable to a wide range of models. Motivated by this, we develop Identical Initialization (IDInit), a sufficiently robust, universal, and fast-converging approach on the identity matrix. Empirical results on a variety of benchmarks show that IDInit is universal to various network types, and practically useful with good performance and fast convergence.

1 INTRODUCTION

Deep Neural Networks (DNNs) have attracted significant attention due to their versatility in various applications. To obtain a well-conditioned training, a suitable initialization is important (Arpit et al., 2019; Huang et al., 2020; Sutskever et al., 2013). Common initialization methods include Xavier (Glorot & Bengio, 2010) and Kaiming initialization (He et al., 2015). Later, dynamical isometry (Saxe et al., 2014) is widely used for building stable starting status for a very deep network (Mishkin & Matas, 2016; Burkholz & Dubatovka, 2019), and even can stably train 10000-layered networks (Xiao et al., 2018). However, despite the successes, these aforementioned methods are unsuitable for residual blocks without Batch Normalization since calculating an improper scale on data-flow (see Sec. 4.2 and Sec. C.2). Addressing this issue, Bachlechner et al. (2021); Hardt & Ma (2017) proposed to stabilize the training of residual blocks according to dynamical isometry mechanism by transiting identity via multiplying residual stem with 0. Nevertheless, the identity-maintaining methods can be only applied on residual networks, and some of them even require auxiliary components to stabilize model training (Blumenfeld et al., 2020; Bachlechner et al., 2021; Zhao et al., 2021), leading to lost generality to other network structures. To overcome the above problems, in this paper, we propose a stable, general, and fast-converging initialization approach, based on the identity matrix.

Motivation on Identity Matrix. As mentioned above, implementing identity transition can naturally correspond to the isometric mechanism that is beneficial for fast convergence and improving performance (Bachlechner et al., 2021). To maintain this transition of both residual and non-residual modules, the identity matrix is a potential solution. In detail, consider i-th block in a DNN,

$$x^{(i+1)} = (r + \prod_{j=1}^{m} \theta^{(i,j)}) x^{(i)}, \tag{1}$$

where m is the number of weights in a block, $\theta^{(i,j)}$ means j-th weight in the i-th block, and $r \in \{0, I\}$ indicates whether Eq. (1) denotes a residual layer. Usually when $r = 0$ and $m = 1$, Eq. (1) is

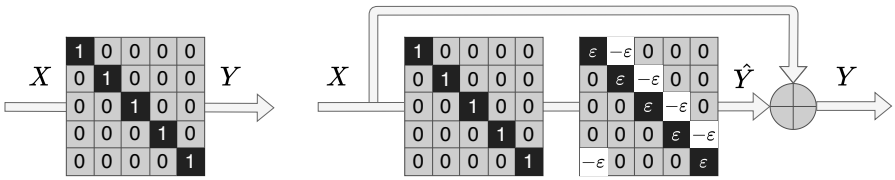


Figure 1: A simple case of IDInit on a residual and non-residual network layer. ε is set to $1e-6$, therefore, $\hat{Y} \approx 0$. Y of both left and right sub-figures is equal to X.

a non-residual layer. Under this condition, to achieve the identity transition, namely $x^{(i+1)} = x^{(i)}$, $\theta^{(i,1)} = I$ is the unique solution. When $r = I$ and $m \geq 2$, Eq. (1) denotes a residual layer. At this moment, setting all $\{\theta^{(i,j)}\}_{j=1}^{m}$ to I is not feasible. Therefore, it is usual to set the last weight $\theta^{(i,m)}$ to 0 to maintain identity (Zhang et al., 2019; Zhao et al., 2021). In all, applying an identity matrix as an initial status to maintain identity, while without loss of the generality, is necessary and feasible.

Identical Initialization (IDInit). In this paper, we introduce the identity matrix into designing a novel and practical initialization, named Identical Initialization (IDInit). A simple case of IDInit is shown in Figure 1. For a non-residual condition, squared identity matrices can easily achieve identity transition, however, the most general condition, non-squared matrices cannot (Vaswani et al., 2017). To adapt the non-squared situation, we modify the identity matrix by maintaining signal variance as described in Sec. 3.2. For a residual condition, a dead neuron problem will be caused by directly setting the last weight in a residual stem to 0 (Zhang et al., 2019; Zhao et al., 2021). Addressing this, we simply select some elements to an extremely small numerical value ε to increase trainable neurons as in Figure 1. We also observe that an identical convolution layer that maintains identity transit degrades performance dramatically. Tackling the issue, we propose a modest change to an identical convolution layer by fusing spatial information, leading to significant improvement. Moreover, we find model performance is uncertain with random initialization whose values are sampled from a probability distribution (Glorot & Bengio, 2010; He et al., 2015). IDInit is exactly able to handle this situation for its determinacy that is unique, and uncorrelated with samplers.

To our best knowledge, it is the first trial to put identical-like initialization into practice. Previous work, Bartlett et al. (2019), has used the identity matrix as a weight with excessive attention on approximate analysis, resulting in imprecise conclusions for the gap between reality and theory. Additionally, this work addresses only relatively simple scenarios without an activation function and ignores momentum in stochastic gradient descent (SGD), which can optimize critical points to a degree. And this method is incompatible with ResNets without Batch Normalization (Ioffe & Szegedy, 2015), since signals will explode if all weights are set identically. Motivating from some of the observations and addressing shortcomings, we provide a simple, practical, and efficient initialization method.

2 Background and Related Work

Dynamical Isometry. Give an L-layer network with blocks formulated by Eq. (1), $x^{(0)}$ is input and $x^{(L)}$ is the L-th layer's output. Assuming signal magnitude (e.g., $\sigma^2(x^{(i)})$) of each layer changing in 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. Introduced from mean-field theory (Pennington et al., 2017; 2018), dynamical isometry is comparably reasonable mechanism to measure models' trainablity. To utilize this paradigm, it usually considers the input-output Jacobian

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

whose mean squared singular value is χ. (Pennington et al., 2017) and (Bachlechner et al., 2021) show that $\chi > 1$ indicates the model 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 exponentially vanish. $\chi = 1$ is a critical line of initialization, avoiding vanishing or exploding gradients. Using the stable isometry condition can provide sufficient robust training process for networks (Gilboa et al., 2019; Poole et al., 2016; Yang & Schoenholz, 2017).

Related Initialization. Xavier (Glorot & Bengio, 2010) and Kaiming initialization (He et al., 2015) is classical initialization. Specially for residual networks efficiency, Hardt & Ma (2017) theoretically demonstrates that network training benefits from keeping identity. Then, maintaining identity transit turns into a promising way to further improve model performance. Fixup (Zhang et al., 2019) and ZerO (Zhao et al., 2021) both set residual stem to 0 (not residual connections) to guarantee identity of signals, thereby initializing ResNets successfully. 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.

3 IDENTICAL INITIALIZATION (IDINIT)

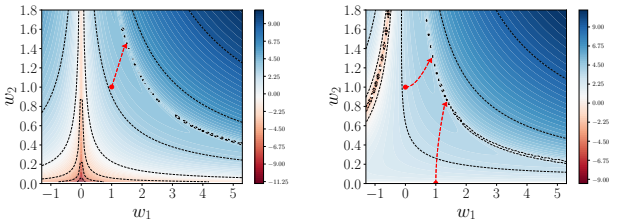
In this section, we first use a descriptive example to elaborate on the mechanism of IDInit for fast-convergence. Then, we describe the way to maintain the stability for non-square identity-like matrices. Next, we propose a zero preserving scheme to solve the dead neuron problem in identity transition. At last, we use a simple but efficient reshaping strategy to significantly enhance convolution performance.

3.1 TOY EXAMPLE

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

$$x^{(L)} = (r + w^{(2)}w^{(1)})^L x^{(0)}, \quad (3)$$

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. (3) 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 delve into a gradient update analysis. Following gradient descent, w_1 can be updated by



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

Figure 2: 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 does not also pass the poor regions around $w^{(1)} = -1, w^{(2)} = 1$ and converge to the solution $w^{(1)}w^{(2)} = 1.19$.

$$w^{(1)} = w^{(1)} - \lambda L w^{(2)} x^{(0)} (r + w^{(2)}w^{(1)})^{L-1} \partial_x R(x)|_{x=x^{(L)}}, \quad (4)$$

where R means the loss function, and λ is a learning rate. As $w^{(1)}$ and $w^{(2)}$ are equivalent in Eq. (3), $w^{(2)}$ can be updated similar to Eq. (4). When $w^{(1)} = 1$, updates is required less than 1. Therefore, the learning rate is constrained to

$$\begin{cases} \lambda \propto L^{-1}, & \text{if non-residual,} \\ \lambda \propto L^{-1}(1 + w^{(2)})^{L-1}, & \text{if residual.} \end{cases} \quad (5)$$

For 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

$$w^{(2)} = -\lambda L w^{(1)} x^{(0)} \partial_x R(x)|_{x=x^{(L)}}, \quad (6)$$

and will converge with a learning rate that is polynomial in the depth L of the network. We plot the training dynamics in Figure 2, and use this simple example to illustrate the mechanism of IDInit, which is always a well-conditioned position for training.

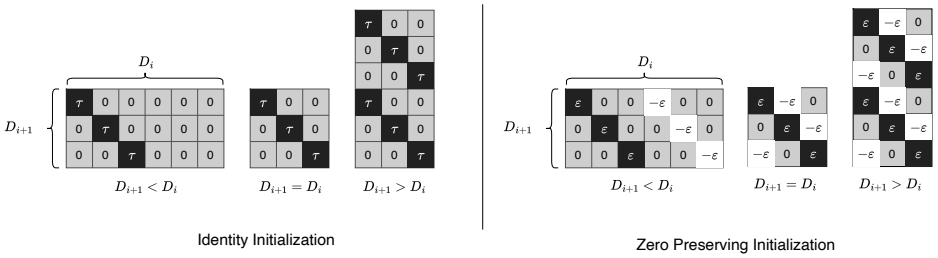


Figure 3: Illustration of IDInit for fully-connected layers.

3.2 Identical Fully-Connected Layers

A standard identity matrix can naturally satisfy identity transition as illustrated in Figure 1. However, in a non-square situation, this natural advantage is lost. To address this problem, we give a small modification 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 $\theta^{(i)} \in \mathbb{R}^{D_{i+1} \times D_i}$ to

$$\theta^{(i)}_{m,j} = \begin{cases} \tau, & \text{if } m \equiv j \pmod{D_i}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The initialization formulated as Eq. (7) is termed as IDI_τ, where IDI means the identical initialization function, and τ is calculated by considering activation function, e.g., $\tau_{ReLU} = \frac{1}{2}$ and $\tau_{sigmoid} = 1$ for ReLU and sigmoid activation functions respectively. As in Figure 3, setting $\tau = 1$ can form IDI_1 initialization.

Trainability Analysis. Magnitude of signals transiting in layers is usually related to stability, and the variance of signals can be a good indicator of the magnitude according to the variance-control mechanism. Obviously, IDI_τ can maintain forward signal variance. Therefore, we explore the trainability from the angle of the gradient backward procedure. Prior to that, consider a network $x^{(L)} = f(x|\Theta) = \theta^{(L-1)}\theta^{(L-2)} \dots \theta^{(0)} x^{(0)}$, the gradient relationship can be formulated as

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i)}} = \theta^{(i)\mathrm{T}} \frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i+1)}} = \theta^{(i)\mathrm{T}} \theta^{(i+1)\mathrm{T}} \dots \theta^{(L-1)\mathrm{T}} \frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L)}}, \qquad \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \theta^{(i)}} = \frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i+1)}} x^{(i)\mathrm{T}}, \tag{8}$$

where $x^{(i)} \in \mathbb{R}^{D_i}$, $\theta^{(i)} \in \mathbb{R}^{D_{i+1} \times D_i}$ and $\boldsymbol{\mathcal{L}}$ denotes a loss of training. Since $\forall i, D_{i+1} = D_i$, the training process will be stable as that gradients transit identically. Here, we pay attention to non-square conditions. $\tau = 1$ for there is no activation function.

If $\forall i, D_{i+1} < D_i$, then $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i)}}$ ($i \in \{1, 2, \dots, L-1\}$) contains elements of 0, which indicates that $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \theta^{(i)}}$ must contain 0 values, causing failed updating. However, this is a fake dead phenomenon, since all weights will be trained after several updating steps. At the first training step, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \theta^{(L-1)}} = \frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L)}} x^{(L-1)\mathrm{T}}$. For both $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L)}}$ and $x^{(L-1)}$ not definitely containing 0 values, $\theta^{(L-1)}$ can be updated normally and deviates from identity-like form. Therefore, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L-1)}}$ can get normal updating in the next step. As a result, all weights can be updated normally.

If $\forall i, D_{i+1} > D_i$, then the variance of gradients will increase from layer to layer. However, this will not make trouble for training. For $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i)}}$, its variance $\sigma^2(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i)}}) \approx \frac{D_{i+1}}{D_i} \sigma^2(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(i+1)}})$. Therefore, the variance of $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(0)}}$ can be calculated as

$$\sigma^2(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(0)}}) = \frac{D_1}{D_0} \frac{D_2}{D_1} \dots \frac{D_L}{D_{L-1}} \sigma^2(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L)}}) = \frac{D_L}{D_0} \sigma^2(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial x^{(L)}}). \tag{9}$$

According to Eq. (9), the gradient will not explode in the backward procedure.

3.3 Identical Residual Blocks

At present, residual blocks become the most popular module in almost all the neural network (e.g., Mixers (Liu et al., 2021a; Tolstikhin et al., 2021), Convolutions (He et al., 2016; Zhang et al., 2019),

and Transformers (Liu et al., 2021b; Vaswani et al., 2017)). A residual block is usually constructed with a residual connection and several transformations in the residual stem. Following Eq. (1), a simple residual case with two parameter matrices can be formulated as

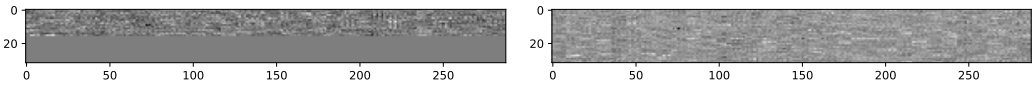
$$x^{(i+1)} = (I + \theta^{(i,0)}\theta^{(i,1)})x^{(i)}, \tag{10}$$

where $x^{(i)}$ 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.

Recent research (Zhang et al., 2019; Zhao et al., 2021) directly sets (i) the last transformation in the residual stem to 0, i.e., $\theta^{(i,0)} = 0$, thereby maintaining identity as

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

(ii) the last classification layer of the network to 0. Although the two settings work, no interpretation is provided for the trainability. And we consider the effect of (i) is from dynamical isometry, since the mean squared singular value χ of this identity-control network's input-output Jacobian is 1. As for the setting (ii), the last layer can be trained with arbitrary values when transiting identity in networks, according to the trainability analysis in Sec. 3.2. Therefore, if all values are set to 0, it can obtain non-zero gradients of a suitable magnitude. Since the settings (i) keeps identity transition and (ii) helps to improve training stability, we also consider adopting them to our initialization. However, while there is no problem with the setting (ii), the setting (i) contradicts the proposed identical initialization as generating a dead neuron problem.



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

Figure 4: The last weight in a residual block of a trained ResNet. More than half elements in (a) are not trained, which is known as the dead neuron. By contrast, IDIZ_{1e-6} successfully solve the dead neuron problem and make all the elements in (b) trainable.

Dead Neuron Problem. Fixup (Zhang et al., 2019) only uses a multiplier of 1 after $\theta^{(i,0)} = 0$, thereby obtaining non-zero gradients. However, following Batch Normalization whose multiplier is 0, $\theta^{(i,0)}$ always acquire gradients with 0 values, known as the dead neuron problem, causing failed weight updating.

To overcome the problem, we generate small values on $\theta^{(i,0)}$ to assist in training. Recall the goal of setting (i) that outputs 0. Therefore, we consider building a calculation to get the expectation and variance of outputs approaching to 0. Prior to that, we consider 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 $\theta^{(i,0)} \in \mathbb{R}^{D_{i+1} \times D_i}$ as an instance. At first, we initialize $\theta^{(i,0)}$ with IDI_ε. Then consider two cases: (i) if $D_{i+1} < D_i$, setting $\theta^{(i)}_{:,D_{i+1}+1:D_i}$ with $\text{IDI}_{-\varepsilon}$; (ii) if $D_{i+1} \geq D_i$, set $\theta^{(i)}_{m,j} = -\varepsilon$, when $m\%D_i = j - 1$. Therefore, we can obtain a variance of 0 by setting ε to a small value. This method is termed as IDIZ_ε. In this paper, we set $\varepsilon = 1e - 6$ everywhere. As shown in Figure 4, IDIZ_{1e-6} successfully initializes the last weight in a residual block.

3.4 Identical Convolutional Layers

Besides the fully-connected layer, the convolutional layer is another important structure in deep neural networks. For constructing a general initialization, we also explore an initialization pattern

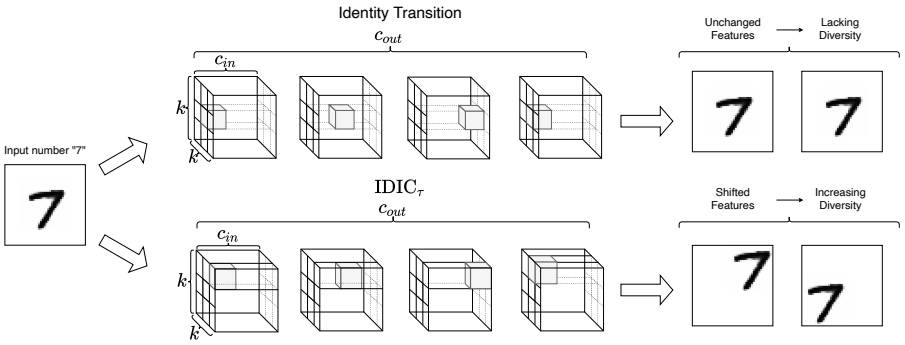


Figure 5: A case of number "7" on Identical Convolution Layer. The upper sub-figure maintains identity transition. The under sub-figure is IDIC$_\tau$ initialization that shifts features for increasing diversity. More feature diversity from IDIC$_\tau$ is beneficial for improving model performance.

for convolution with identity transition. In this part, we consider a convolution layer formulated as

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{C}} \otimes \boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{B}}, \tag{13}$$

where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{h \times w \times c_{in}}$ is an input tensor, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{h' \times w' \times c_{out}}$ is a output tensor, $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{k \times k \times c_{in} \times c_{out}}$ is a convolutional kernel, and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{h' \times w' \times c_{out}}$ is a bias and usually set to 0. h, w and h', w' denote height and width of the input and the output, respectively. c_{in} and c_{out} are channels of input and output separately. k means convolutional kernel size.

Degeneration from Convolution Identity Transition. As shown upper sub-figure in Figure 5, similar to an identity matrix, a convolution layer can transit identity by setting 0-filled $\boldsymbol{\mathcal{C}}$ as

$$\text{IDI}_1(\boldsymbol{\mathcal{C}}_{n,n,:,:}), \tag{14}$$

where $n \in \mathbb{N}^+$ and $k = 2n + 1$. As a convolutional kernel window size, k is usually an odd number. After Eq. (14), $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}}$ if $c_{in} = c_{out}$. $\boldsymbol{\mathcal{Y}}$ will under-sample and over-sample on $\boldsymbol{\mathcal{X}}$ along channel when $c_{in} > c_{out}$ and $c_{in} < c_{out}$ respectively. As keeping identity is usually considered an efficient way to improve model performance, however, we find that this setting can lead to a fatal performance degeneration (see Sec. 4.3).

Inspired by Han et al. (2020) that enhances model performance by increasing feature diversity, we propose to fuse spatial information by simply reshaping a matrix initialized IDI$_\tau$. Specifically, consider one inner product part of a convolution on feature patch $Y^{(patch)} \in \mathbb{R}^{c_{out}}$ as

$$Y^{(patch)} = CX^{(patch)}, \tag{15}$$

where $C \in \mathbb{R}^{c_{out} \times kkc_{in}}$ is the convolutional kernel, and $X^{(patch)} \in \mathbb{R}^{kkc_{in}}$ is input patch. By initializing C with IDI$_\tau$, then 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. As shown in under sub-figure of Figure 5, spatial features are shifted, thereby increasing feature diversity. We utilize IDIC$_\tau$ to denote such a reshaping process. In addition, we use IDIZC$_\varepsilon$ to represent the similar operation on IDIZ$_\varepsilon$.

We conclude the whole IDInit as: (1) **Non-Residual Networks.** Directly applying IDI$_\tau$ and IDIC$_\tau$ to all the fully-connected and convolutional layers, respectively; (2) **Residual Networks.** (i) Applying IDI$_\tau$ and IDIC$_\tau$ to all the fully-connected and convolutional layers, respectively; (ii) Applying IDIZ$_\varepsilon$ and IDIZC$_\varepsilon$ to the fully-connected and convolutional layers in the last position of residual blocks, and the position of last classification layer.

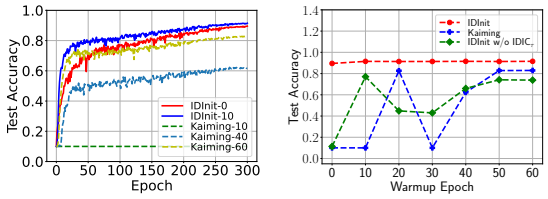
4 Experiment

In this section, we design a set of experiments to validate the proposed IDInit. To begin with, we conduct experiments on non-residual convolution and residual convolution in Sec. 4.1 and Sec. 4.2 respectively. Then we implement an ablation experiment in Sec. 4.3 to show the effect of the proposed two modifications in Sec. 3. Later we analyse the determinacy of initialization in Sec. 4.4. At

last, we employ experiments on tasks large-scale datasets ImageNet and WuDaoCorpora in Sec. 4.5 and Sec. 4.6 separately. We also analyse the variance amplification in Sec. C.2 in Appendix.

4.1 Validation on Non-Residual Convolution

We use this experiment to show the hardness of classical Kaiming initialization for training. This experiment is conducted on Cifar10. We use nine convolutional layers named AllConv (Springenberg et al., 2015). We show the structure of AllConv in Table 3 in the appendix. 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.



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

Figure 6: Results of AllConv on Cifar10. The number behind the initialization denotes the warm-up epochs.

Results are shown in Figure 6, without a warm-up strategy which is a strong trick for training, both Kaiming and IDInit w/o $\mathrm{IDIC}_\varepsilon$ fail to train the model. By contrast, our initialization can train AllConv and maintain the highest performance in all the situations, showing a strong effect on robustness and performance. As IDInit w/o $\mathrm{IDIC}_\varepsilon$ performs awfully, we demonstrate the degeneration from identity transition mentioned in Sec. 3.4 is relieved by our new reshaping strategy.

4.2 Validation on Residual Convolution

In this experiment, we validate the proposed initialization with 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. To verify the robustness, we use two settings, namely w/ and w/o BN. For analysing 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.

Table 1: Results on Cifar10.

		56 Layer (SGD/Adam)		110 Layer (SGD/Adam)	
Model	Init.	Acc.	Epochs to 80% Acc.	Acc.	Epochs to 80% Acc.
ResNet w/ BN	Zero γ	$92.32_{\pm 0.19}$/$87.37_{\pm 0.43}$	$57_{\pm 7}$/$63_{\pm 4}$	$93.07_{\pm 0.28}$/$88.30_{\pm 0.31}$	$36_{\pm 2}$/$56_{\pm 7}$
	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	$\mathbf{93.41}_{\pm 0.10}$/$\mathbf{90.01}_{\pm 0.32}$	$\mathbf{26}_{\pm 1}$/$\mathbf{34}_{\pm 1}$	$\underline{94.04}_{\pm 0.24}$/$\underline{90.53}_{\pm 0.10}$	$\mathbf{27}_{\pm 1}$/$\mathbf{36}_{\pm 2}$
ResNet w/o BN	Kaiming	Diverged/$63.84_{\pm 0.90}$	- / -	Diverged/$16.45_{\pm 1.45}$	- / -
	IDInit	$91.53_{\pm 0.21}$/$86.29_{\pm 0.40}$	$61_{\pm 1}$/$58_{\pm 2}$	$91.51_{\pm 0.14}$/$87.20_{\pm 0.30}$	$57_{\pm 2}$/$49_{\pm 1}$

Results are shown in Table 1. The proposed IDInit can achieve the least epochs to reach 80% accuracy in all settings, which shows a good convergence ability. In the w/ BN setting, IDInit consistently performs well compare with baselines. In the w/o BN setting, Kaiming fails to train the models with SGD optimizer for calculating improper scales for data-flow. Even though using the adaptive Adam optimizer, Kaiming still cannot derive a normal results. By contrast, IDInit can work normally in both optimizers and derive comparably reasonable results.

4.3 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. We run four times following settings: (i) IDInit w/o IDIC_τ and w/o $\mathrm{IDIZC}_\varepsilon$; (ii) IDInit w/o IDIC_τ; (iii) IDInit w/o $\mathrm{IDIZC}_\varepsilon$; (iv) IDInit. 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.

As shown in Figure 7, by applying the identity matrix directly, (i) obtains the lowest accuracy of 87% among all cases. Regarding to results of (ii) and (iii), both the two settings can make significantly improvement of nearly 5.6% and 3.2% from (i), respectively. And IDIC_τ can make a deeper effect than $\mathrm{IDIZC}_\varepsilon$. Equipping IDIC_τ and $\mathrm{IDIZC}_\varepsilon$, IDInit will improve performance further, which demonstrates our modification is efficient and practical.

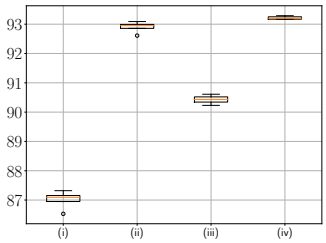


Figure 7: Ablation experiment with ResNet-20 on Cifar10.

4.4 Determinacy Analysis on Text Classification

We conduct this experiment to show random initialization can cause unacceptable uncertainty measured with standard derivation, and IDInit can solve this problem for the inherent determinacy that seeds will not change initial weights when model is determined. We implement text classification on SST2 and SST5 (Socher et al., 2013) 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 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. Fixing other hyper-parameters except for the initialization, we run all the random initialization for 5 times.

Table 2: Results of TextCNN and TextRNN on SST2 and SST5. 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. Std values larger than 1.0 are marked in red.

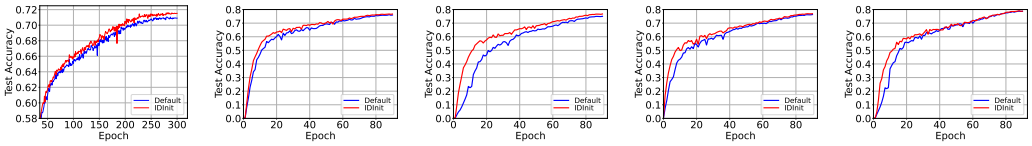
Datasets	Init.	$\mathrm{TextCNN}_{\mathrm{G/W}}$	$\mathrm{TextRNN}_{\mathrm{G/W}}$	$\mathrm{Transformer}_{\mathrm{G/W}}$
SST2	Default	$81.40_{\pm 0.66}/84.56_{\pm 0.43}$	$81.69_{\pm 0.30}/84.29_{\pm 0.70}$	$80.97_{\pm 1.20}/83.36_{\pm 0.76}$
	Orthogonal	$82.24_{\pm 0.44}/84.37_{\pm 0.38}$	$81.86_{\pm 0.55}/84.61_{\pm 0.78}$	$82.22_{\pm 0.87}/83.99_{\pm 0.23}$
	IDInit	**83.14/86.05**	**83.14/86.22**	**82.66/85.13**
SST5	Default	$44.68_{\pm 0.88}/46.15_{\pm 0.62}$	$44.27_{\pm 0.88}/47.04_{\pm 0.48}$	$41.81_{\pm 1.17}/44.02_{\pm 1.27}$
	Orthogonal	$44.91_{\pm 0.81}/46.76_{\pm 0.68}$	$44.61_{\pm 1.18}/46.13_{\pm 0.79}$	$43.01_{\pm 1.61}/44.92_{\pm 1.52}$
	IDInit	**45.66/48.05**	**47.12/48.83**	**45.22/46.24**

As shown in Table 2, all the initialization methods can work normally. However, default random initialization obtains the lowest accuracy in both SST2 and SST5. Orthogonal initialization only performs slighter better than IDInit with TextRNN on SST2, and may cause a bad result as 42.35% with TextCNN on SST5. On the contrary, the proposed IDInit can achieve the highest accuracy in almost all the conditions, which shows the sufficient efficiency of IDInit. Furthermore, as a deterministic method, IDInit offers the ability to develop a fully deterministic training framework as well as improve performance for easy-to-analyze structures.

4.5 Image Classification on ImageNet

In this experiment, we use ViT-B/32 (Dosovitskiy et al., 2021), ResNet-50/152 (RN-50/152) and Se-ResNet-50 (SRN-50) as backbones on ImageNet. 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, the optimizer is SGD with a learning rate 1e-1 and a weight decay 1e-4. The epoch for training is 90. 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 switch probability is 0.5 and a label smoothing with 0.1.



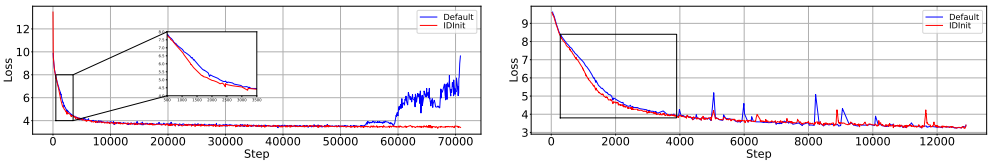
(a) ViT-B/32 (b) RN-50 (Adamw) (c) RN-50 (d) SRN-50 (e) RN-152

Figure 8: 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.

Results are shown in Figure 8. On three types of networks, i.e., ViT, ResNet and Se-ResNet, and multiple depths, IDInit always achieve faster convergence and better performance than the baseline. And when training RN-50 with Adamw, the convergence of IDInit is still consistently fast. This experiment shows the good practicability and promising probability of IDInit, which is beneficial to the artificial intelligence community.

4.6 Pre-Training on WuDaoCorpora

Pre-training plays an important role in vaious application. In this experiment, we pre-train GPT with 8 and 96 layers from scratch on the large-scale text dataset WuDaoCorpora (Yuan et al., 2021) which has 200 GB training data and 72 billion Chinese characters. The optimizer is Adam with a learning rate 1e-3 and a weight decay 5e-4. The batch size is 1536. For 8-layer GPT, we run 70k steps. For 96-layer GPT, we run 13k steps. This experiment are conducted on 32 NVIDIA V100s.



(a) 8-layer GPT Loss (b) 96-layer GPT Loss

Figure 9: Results of GPT on WuDaoCorpora. "Default" means the default initialization of models for GPT. IDInit is our identical initialization.

As shown in Figure 9, IDInit achieves faster convergence on both 8 and 96 depths. Additionally, as in Figure 9(a), the random initialization's loss unexpectedly increases sharply at around 54k step, indicating an unstable training. By contrast, IDInit can steadily be trained normally. This experiment shows good convergence and robustness of IDInit, which can benefit pre-training to some extent.

5 Conclusion

In this paper, we introduce an 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 zero 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 robustness, and fast convergence, IDInit is promising to be applicable in practice. In the future, we hope that this identical design can motivate AI community to implement more novel initialization methods.

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.

Rebekka Burkholz and Alina Dubatovka. Initialization of relus for dynamical isometry. In *NeurIPS*, pp. 2382–2392, 2019.

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

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.

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.

Xiao Shi Huang, Felipe Pérez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4475–4483. PMLR, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015.

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.

Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps. In *NeurIPS*, pp. 9204–9215, 2021a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pp. 9992–10002. IEEE, 2021b.

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

Dmytro Mishkin and Jiri Matas. All you need is a good init. In *ICLR (Poster)*, 2016.

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.

Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. The emergence of spectral universality in deep networks. In *AISTATS*, volume 84 of *Proceedings of Machine Learning Research*, pp. 1924–1932. PMLR, 2018.

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.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014.

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.

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, pp. 24261–24272, 2021.

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.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5389–5398. PMLR, 2018.

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

Sha Yuan, Hanyu Zhao, Zhengxiao Du, Ming Ding, Xiao Liu, Yukuo Cen, Xu Zou, Zhilin Yang, and Jie Tang. Wudaocorpora: A super large-scale chinese corpora for pre-training language models. *AI Open*, 2:65–68, 2021.

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.

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 Schäfer, and Anima Anandkumar. Zero initialization: Initializing residual networks with only zeros and ones. *CoRR*, abs/2110.12661, 2021.

Appendix

Table of Contents

A IDInit Details **13**

A.1 Full IDInit Scheme 13

A.2 Implementing IDInit on Attention Layer in Transformer 14

B Detailed Settings of Experiments **14**

B.1 Details of Validation on Non-Residual Convolution Experiment 14

B.2 Details of Validation on Residual Convolution Experiment 14

B.3 Details of Ablation Experiment 15

B.4 Details of Determinacy Analysis on Text Classification Experiment 15

B.5 Details of Image Classification on ImageNet Experiment 15

B.6 Details of Pre-Training on WuDaoCorpora Experiments 15

C Additional Experiments **15**

C.1 Validation on the Linear Structure 15

C.2 Analysis on Variance Propagation 16

A IDInit Details

A.1 Full IDInit Scheme

Here, we show the full IDInit scheme in Figure 10.

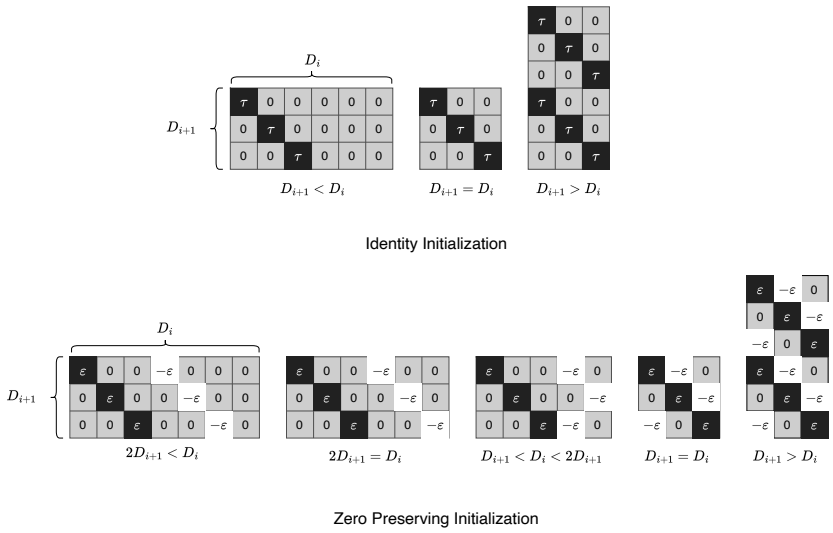


Figure 10: Illustration of IDInit with all conditions.

A.2 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

$$\mathrm{Att}(Q, K, V) = \mathrm{softmax}(\frac{QW^QW^KK}{\sqrt{d}})VW^VW^O, \tag{16}$$

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.

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 DETAILS OF VALIDATION ON NON-RESIDUAL CONVOLUTION EXPERIMENT

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 3. The dataset is Cifar10. 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 $\mathrm{IDIZC}_\varepsilon$ 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 3: 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

B.2 DETAILS OF VALIDATION ON RESIDUAL CONVOLUTION EXPERIMENT

In this experiment, we validate the proposed initialization with 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. To verify the robustness, we use two settings, namely w/ and w/o BN. For analysing 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.

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.

B.4 DETAILS OF DETERMINACY ANALYSIS ON TEXT CLASSIFICATION EXPERIMENT

We also explore performance networks on text classification datasets including SST2 and SST5 (Socher et al., 2013) 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 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. Fixing other hyper-parameters except for the initialization, we run all the random initialization for 5 times. The experiment is conducted on one Nvidia A100.

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 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 switch probability is 0.5 and a label smoothing with 0.1. The experiment is conducted on 4 Nvidia A100.

B.6 DETAILS OF PRE-TRAINING ON WUDAOCORPORA EXPERIMENTS

In this experiment, we conduct experiments on WuDaoCorpora (Yuan et al., 2021) which has 200 GB training data and 72 billion Chinese characters. We use GPT with 8 and 96 layers as backbones. The optimizer is Adam with a learning rate 1e-3 and a weight decay of 5e-4. And the batch size is 1536. We use 20k steps for warm-up. For 8-layer GPT, we run 70k steps. For 96-layer GPT, we run 13k steps. The experiment is conducted on 32 Nvidia V100.

C ADDITIONAL EXPERIMENTS

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

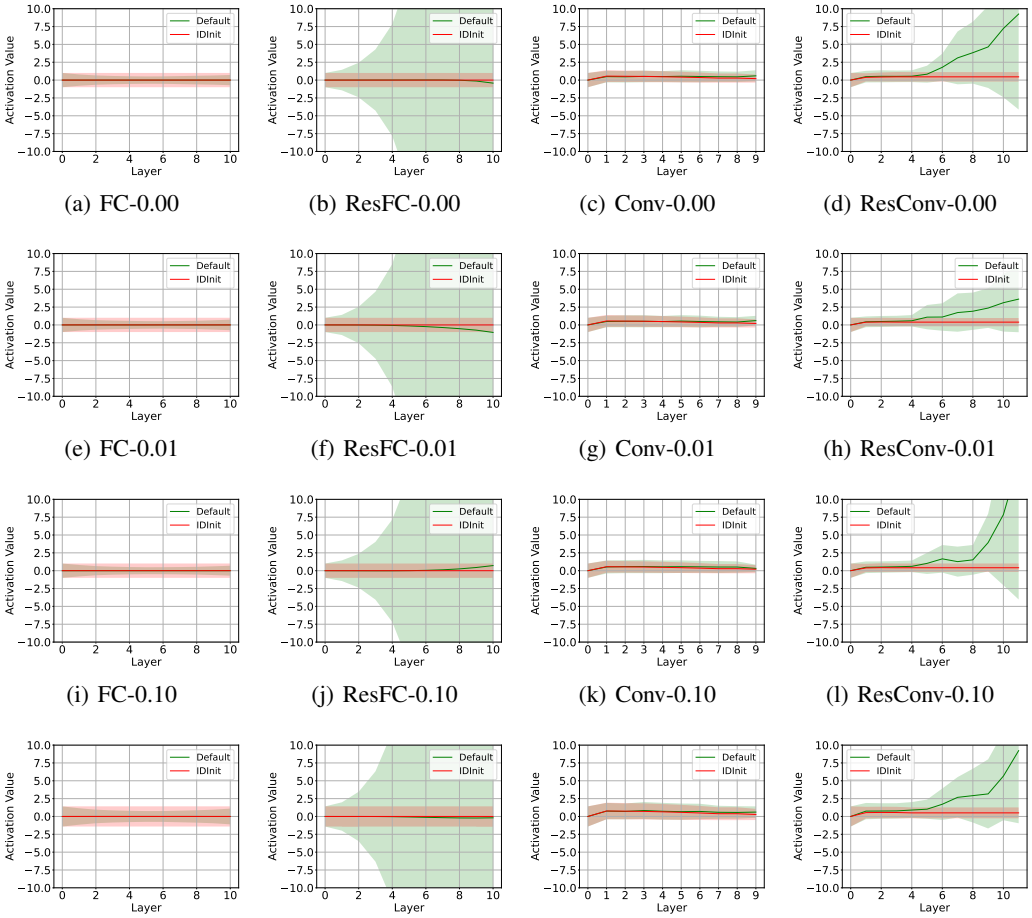
C.1 VALIDATION ON THE LINEAR STRUCTURE

This experiment is conducted on MNIST. We use five linear layers named Liner-5 whose hidden layers are all of dimension 512. Optimizer is SGD with momentum 0.9, weight decay 5e-4, and a learning rate 1e-1. Learning rate scheduler adopts cosine reduction strategy. We run the model in 30 epochs on one Nvidia A100. We both consider Linear-5-tanh and Linear-5-ReLU that consist of Linear-5, and tanh and ReLU activation functions, respectively. The experiment is conducted on one Nvidia A100.

As shown in Table 4, 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.

Table 4: 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**



(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 11: Results of 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 stable and efficient training.

C.2 Analysis on Variance Propagation

Here we conduct an experiment on Cifar10 to demonstrate data-flow will keep stable. We use 4 types 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. B.1; (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 3. For (4), we firstly use a convolution to transfer a image to 16 channel, and then set 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 terms to 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 11. 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 stable and efficient training.