Simplified Reversible Residual Networks

Erland Brandser Olsson[*1]
erland.b.olsson@ntnu.no

Zhirong Yang[1 2]
zhirong.yang@ntnu.no

Abstract

The de facto standard way of training neural networks requires storing activations at each layer in memory, in order to compute gradients in backpropagation. As progress has increasingly relied on stacking more and larger layers, memory consumption has now become a major bottleneck for making frontier models available to everyone. In this work, we take advantage of the Residual Network (ResNet) layer formulation that is now used in many state-of-the-art neural network architectures to recompute the activations at each layer on-the-fly during backpropagation instead of storing them in memory. We propose a modification to the Reversible Residual Network (RevNet) that removes the requirement to replace the ResNet layer network function with two new network functions by setting one of them to the identity function. This simple modification offers a significant advantage: it eliminates the need to change the original ResNet architecture. Our method can work as a drop-in replacement for layers with residual connections such as in ordinary ResNets or Transformers with negligible performance loss while offering significant reductions in memory consumption. With this simple modification, we are able to maintain the same computational cost and ease of use as activation checkpointing, which requires no architectural changes, while leveraging a reversible procedure such that we only need to store the activations from the last layer. We call this method Simplified RevNet and, compared with previous reversible architectures, propose a simpler and more streamlined approach that comes in two variants based on which of F or G in RevNet becomes an identity function. Empirically, we demonstrate performance at practically the same level as the non-reversible counterparts on ImageNet image classification with ResNets and on OpenWebText language modeling with Transformers.

1 Introduction

Neural networks underwent a resurgence around 2010 under the new name *deep learning*, and after many decades of steady progress, the field has now entered a phase of very rapid development [1–4]. This progress has driven major advances in areas such as computer vision and natural language processing; however, the computational demands of training state-of-the-art neural networks have become a significant practical barrier, and graphics processing unit (GPU) memory in particular has become a major bottleneck. The performance gains of modern neural networks over earlier architectures are often attributed to greater network depth [3]. Architectural innovations such as residual networks (ResNets) [5] have been central to this progress, as they provide direct pathways for information and gradients to propagate through deep networks, thereby mitigating vanishing and exploding gradient problems and enabling the training of substantially deeper models. Nearly all neural networks today are trained using the backpropagation algorithm (backprop) [6]. Since

[*] Corresponding Author.
[1] Department of Computer Science, Norwegian University of Science and Technology, Norway
[2] Jinhua Institute of Zhejiang University, China

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

backprop requires storing activations from each layer, the dominant training memory cost is usually associated with backprop rather than parameter storage, even as models grow in depth and parameter count. Despite its high memory cost, backprop remains the default algorithm for training neural networks because it is simple, effective, and hard to beat in practice. A common workaround for the memory bottleneck of backprop is activation checkpointing, where activations are stored only at selected checkpoints, while the activations in between are recomputed during the backward pass. This trades memory for additional computation, since part of the forward pass must be performed again during backprop. Although this can substantially reduce storage compared with standard backprop, the memory cost still generally increases linearly with network depth, making it unsuitable for extremely low-memory settings. To eliminate activation storage entirely, reversible architectures make each layer invertible, so that intermediate activations can be reconstructed during the backward pass, reducing the activation memory cost to $\mathcal{O}(1)$ in principle, since only the final-layer activations must be stored. RevNet [7] was an important step in this direction, as it made the additive structure of residual layers in ResNets reversible and empirically verified that such reversible networks can work as replacements for ResNets. However, to enable inversion, RevNet must replace the residual function in a ResNet layer with two new, typically shallower, functions F and G, and split the input into two equally sized partitions that are fed into two separate branches. A RevNet layer is therefore not a drop-in replacement for a standard ResNet layer, as the architecture has to be adjusted to match the depth and parameter count of an equivalent ResNet. In addition, the coupling between the two new functions F and G, each applied to a different input split, has limited justification from an information-theoretic standpoint. To address these limitations, we propose Simplified RevNet (SRevNet). SRevNet replaces the input in the first RevNet branch with an accumulator $\boldsymbol{x}_a$, and sets one of the two functions to the identity, yielding SRevNet v1 ($G = I$) and SRevNet v2 ($F = I$). Both versions preserve the original ResNet function and avoid input splitting, making them practical drop-in replacements for ResNet-style layers. Our experiments on ImageNet with ResNets and OpenWebText with Transformers show that our method simplifies the implementation of existing reversible designs, making them more practical to deploy without compromising performance or increasing computational overhead.

2 Problem Setup, Motivation, and Related Work

2.1 Backpropagation

The backpropagation algorithm (backprop) [6] is the standard method for training neural networks. More generally, this procedure is known as reverse-mode differentiation: activations from the forward pass are stored, and gradients are computed by working backward, applying the chain rule from the scalar loss to the network input. If layer inputs and outputs are treated as vectors, the backward pass can be expressed as a sequence of vector-Jacobian products (VJPs). Suppose layer t computes

$$\boldsymbol{x}^{(t+1)} = F(\boldsymbol{x}^{(t)}), \tag{1}$$

where F is the layer transformation and $\boldsymbol{w}^{(t)}$ denotes its weight parameters, assumed to be represented as a vector. Let $\bar{\boldsymbol{x}}^{(t)} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{(t)}}$ be the gradient of the loss with respect to the input of layer t. Then

$$\bar{\boldsymbol{x}}^{(t)} = \left(\left(\bar{\boldsymbol{x}}^{(t+1)}\right)^T \frac{\partial \boldsymbol{x}^{(t+1)}}{\partial \boldsymbol{x}^{(t)}} \right)^T = \left(\frac{\partial \boldsymbol{x}^{(t+1)}}{\partial \boldsymbol{x}^{(t)}} \right)^T \bar{\boldsymbol{x}}^{(t+1)}. \tag{2}$$

Once $\bar{\boldsymbol{x}}^{(t+1)}$ is known, the weight gradient can be obtained as

$$\bar{\boldsymbol{w}}^{(t)} = \left(\frac{\partial F(\boldsymbol{x}^{(t)})}{\partial \boldsymbol{w}^{(t)}} \right)^T \bar{\boldsymbol{x}}^{(t+1)}. \tag{3}$$

These VJP formulations will be useful later when deriving the backprop procedure for SRevNet. Backprop is efficient because the input gradient $\bar{\boldsymbol{x}}^{(t+1)}$ is reused to compute gradients layer by layer. Its main cost is memory: standard backprop must store activations to compute the required Jacobians for the backward pass. This is a major bottleneck in modern training and is often described as a memory-reuse problem [8, 9], because activations from many layers must remain available when backprop begins rather than the same memory buffers being reused in other layers. Forward-mode differentiation [10] propagates gradients forward, naturally avoiding activation storage, yet in practice its computational cost is comparable to processing one weight component per forward pass, making

it impractical for large networks with many parameters. Approximate one-pass variants based on forward-mode differentiation and local losses [11, 9, 12] have therefore been explored but do not yet match the accuracy of exact gradient methods in general.

2.2 Residual Networks

Residual connections [5] are critical in training deep networks, as they provide an explicit identity path that allows activations and gradients to bypass one or more sublayers. A plain layer computes $\boldsymbol{x}^{(t+1)} = F(\boldsymbol{x}^{(t)})$, whereas a ResNet layer can be written as

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + F(\boldsymbol{x}^{(t)}), \tag{4}$$

where F typically denotes a block of layers (the original ResNet paper used either Basic or Bottleneck blocks). The input gradient is

$$\bar{\boldsymbol{x}}^{(t)} = \left(\boldsymbol{I} + \left(J_F^{(t+1)}\right)^T\right) \bar{\boldsymbol{x}}^{(t+1)}, \tag{5}$$

where $J_F^{(t+1)} = \partial F(\boldsymbol{x}^{(t)})/\partial \boldsymbol{x}^{(t)}$. For a plain layer, the gradient is $\bar{\boldsymbol{x}}^{(t)} = \left(J_F^{(t+1)}\right)^T \bar{\boldsymbol{x}}^{(t+1)}$. The additive ResNet form is important because both forward and backward pass dynamics decompose into purely additive terms. The upstream gradient $\bar{\boldsymbol{x}}^{(t+1)}$ can now flow directly to layer t, bypassing the adjacent layer rather than being multiplied through its Jacobian. ResNet simplified the weighted shortcut connection used in Highway Networks [13],

$$\boldsymbol{x}^{(t+1)} = \gamma_1 \boldsymbol{x}^{(t)} + \gamma_2 F(\boldsymbol{x}^{(t)}), \tag{6}$$

by removing the weights γ_1 and γ_2, which weight the identity and transformed paths, respectively; in a Highway layer they form a learned gate, e.g. $\gamma_1 = T(\boldsymbol{x}^{(t)})$, $\gamma_2 = 1 - T(\boldsymbol{x}^{(t)})$ with $T(\boldsymbol{x}^{(t)}) \in [0, 1]$. The input gradient is

$$\bar{\boldsymbol{x}}^{(t)} = \left(\gamma \boldsymbol{I} + (1-\gamma)\left(J_F^{(t+1)}\right)^T\right) \bar{\boldsymbol{x}}^{(t+1)}, \tag{7}$$

with $\gamma = T(\boldsymbol{x}^{(t)})$. Unrolling the two recursions above shows that multiplicative γ factors accumulate with depth, producing term scalings as $\propto \gamma^L$ and $\propto (1-\gamma)^L$ over L layers. A follow-up analysis [14] highlights that the success of ResNet stems precisely from removing these weights, which avoids the repeated γ scaling that can adversely affect training through vanishing or exploding gradients. That is why ResNet is now considered the default choice and is used in almost all architectures, such as the Transformer [15], where F alternates between an attention and a feed-forward block.

2.3 Reversible Networks

Reversible Residual Network (RevNet) [7] made the additive update in a ResNet layer invertible so that additions in the forward pass are undone by subtractions in the backward pass. To make the activations reversible, RevNet introduced a two-step procedure applied to two partitions $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ split from the input $\boldsymbol{x}$ defined as

$$\begin{aligned} \boldsymbol{y}_1 &= \boldsymbol{x}_1 + F(\boldsymbol{x}_2) \\ \boldsymbol{y}_2 &= \boldsymbol{x}_2 + G(\boldsymbol{y}_1), \end{aligned} \tag{8}$$

where the original ResNet function must be replaced by two shallower networks F and G, preventing direct drop-in use. The input to the next layer becomes $\mathrm{Concat}([\boldsymbol{y}_1, \boldsymbol{y}_2])$. The inverse transformation $\boldsymbol{x}_2 = \boldsymbol{y}_2 - G(\boldsymbol{y}_1)$ and $\boldsymbol{x}_1 = \boldsymbol{y}_1 - F(\boldsymbol{x}_2)$ reconstructs each layer's activations on the fly during backprop. Although zero intermediate activation storage is theoretically possible, practical limits exist because standard block designs are not strictly invertible. For example, ResNet Basic and Bottleneck blocks often change feature size through expansion, reduction, or downsampling, and they may use irreversible activation functions. Because we build ResNets like this, reversible networks often end up behaving as activation checkpointing, where we use backprop within the F block but avoid storing checkpoints. The Reformer [16] extends RevNet to the Transformer [15] by duplicating features along the feature dimension before the split, but this remains a Transformer-specific workaround rather than a general-purpose reversible solution.

Neural ODEs [17] reinterpret ResNets as Euler discretizations of a continuous residual rate and replace discrete layers with an ODE-driven rate of change, enabling $\mathcal{O}(1)$ memory training via

the adjoint sensitivity method, but in practice they often suffer from degraded performance, slower training, and numerical instability. Momentum ResNet (MResNet) [18] is a neural-ODE-inspired momentum variant of ResNet. Since ResNet and RevNet correspond, in ODE terms, to first-order ODEs and may therefore fail to represent higher-order maps (e.g., $x \mapsto -x^3$ in $\mathbb{R}$), MResNet lifts ResNet to a second-order interpretation by adding a velocity state $\boldsymbol{v}^{(t)}$ and updating

$$\begin{aligned} \boldsymbol{v}^{(t+1)} &= \gamma\, \boldsymbol{v}^{(t)} + (1-\gamma)\, F(\boldsymbol{x}^{(t)}) \\ \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} + \boldsymbol{v}^{(t+1)}, \end{aligned} \tag{9}$$

which is a semi-implicit Euler (Verlet) discretization [19] of the second-order ODE $\epsilon \frac{\partial^2 \boldsymbol{x}(t)}{\partial t^2} + \frac{\partial \boldsymbol{x}(t)}{\partial t} = F(\boldsymbol{x}(t))$ with $\epsilon = 1/(1-\gamma)$. The momentum constant $\gamma \in [0, 1]$ then interpolates between a first-order ODE when $\gamma \to 0$ and a second-order ODE when $\gamma \to 1$. MResNet is reversible via $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t+1)} - \boldsymbol{v}^{(t+1)}$ and $\boldsymbol{v}^{(t)} = \big(\boldsymbol{v}^{(t+1)} - (1-\gamma)F(\boldsymbol{x}^{(t)})\big)/\gamma$. However, MResNet's goal is not merely to fix RevNet's drawbacks but to model richer ODE dynamics through a more expressive architecture; in that sense it is a specialized ODE model. Surprisingly, we found that MResNet uses a branching similar to SRevNet v1 ($G = I$). Yet although the interpolation via γ is a well-known trick in the ODE literature, which often operates in continuous-time settings rather than discrete neural network settings, we find that the momentum term induces depth-dependent scaling that complicates backprop and causes numerical reconstruction errors, analogous to Highway Networks.

3 Method

3.1 Simplified RevNet

We now introduce Simplified Reversible Residual Network (Simplified RevNet or SRevNet), a simple and practical reversible adaptation of a ResNet layer $\boldsymbol{y} = \boldsymbol{x} + F(\boldsymbol{x})$. The goal of SRevNet is to make a ResNet layer reversible without changing the ResNet architecture, that is, without changing F. A RevNet layer cannot be used as a drop-in replacement for a ResNet layer because it requires replacing the original F function with two heuristically chosen shallower functions, splitting the input into two halves and processing them in a coupled way. We therefore regard RevNet as a practical workaround rather than an ideal reversible adaptation of a ResNet layer. The key insight is that we can set F or G in Eq. (8) in RevNet to an identity function, and, with the help of an auxiliary variable $\boldsymbol{x}_a$ (set to zero before the first layer), we can derive two simplified versions of a RevNet layer as follows:

SRevNet v1:

$$\begin{aligned} \boldsymbol{y}_a &= \boldsymbol{x}_a + F(\boldsymbol{x}) \\ \boldsymbol{y} &= \boldsymbol{x} + \boldsymbol{y}_a \end{aligned} \tag{10}$$

SRevNet v2:

$$\begin{aligned} \boldsymbol{y}_a &= \boldsymbol{x}_a + \boldsymbol{x} \\ \boldsymbol{y} &= \boldsymbol{x} + G(\boldsymbol{y}_a) \end{aligned} \tag{11}$$

SRevNet v1 in Eq. (10) corresponds to setting G to the identity and rewriting $\boldsymbol{x}_a = \boldsymbol{x}_1$ and $\boldsymbol{x} = \boldsymbol{x}_2$ in the RevNet formulation in Eq. (8). Similarly, SRevNet v2 in Eq. (11) corresponds to setting F to the identity. The only additional cost is instantiating $\boldsymbol{x}_a$, whose shape matches the input $\boldsymbol{x}$, which is negligible in practice. To invert the activations at the layer, we apply the following equations:

SRevNet v1:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{y} - \boldsymbol{y}_a \\ \boldsymbol{x}_a &= \boldsymbol{y}_a - F(\boldsymbol{x}) \end{aligned} \tag{12}$$

SRevNet v2:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{y} - G(\boldsymbol{y}_a) \\ \boldsymbol{x}_a &= \boldsymbol{y}_a - \boldsymbol{x} \end{aligned} \tag{13}$$

Compared with a standard ResNet layer $\boldsymbol{y} = \boldsymbol{x} + F(\boldsymbol{x})$, SRevNet v1 can be written as

$$\boldsymbol{y} = \boldsymbol{x} + \widetilde{F}(\boldsymbol{x}), \qquad \widetilde{F}(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}_a. \tag{14}$$

Thus, SRevNet v1 modifies the skip connection $\widetilde{F}(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}_a$, where the change to F is minimum, while there is accumulated change in $\boldsymbol{x}_a$ added to $F(\boldsymbol{x})$. SRevNet v2 can be written as

$$\boldsymbol{y} = \boldsymbol{x} + \widetilde{G}(\boldsymbol{x}), \qquad \widetilde{G}(\boldsymbol{x}) = G(\boldsymbol{x} + \boldsymbol{x}_a). \tag{15}$$

Thus, SRevNet v2 modifies the input to the network by adding $\boldsymbol{x}_a$, that is, by using $\widetilde{G}(\boldsymbol{x}) = G(\boldsymbol{x} + \boldsymbol{x}_a)$, while the skip connection is unchanged. This highlights that in SRevNet v2, $\boldsymbol{x}_a$ is injected into the learnable network function, whereas in SRevNet v1 it is added directly to the output rather than being passed through the learnable network function. In this sense, SRevNet v2 preserves the familiar ResNet paradigm output = input + transform more directly than SRevNet v1. Unlike

RevNet, in both cases, SRevNet allows the original ResNet module—denoted F in v1 and G in v2—to be used without modification. This is the main advantage over RevNet. In addition to being more practical, we believe the SRevNet formulations have a more intuitive interpretation. We will clarify this when we compare the gradients below. The full backprop procedures for SRevNet v1 and SRevNet v2 are given in Algorithms 1 and 2.

Algorithm 1 Simplified BlockReverse v1

function BlockReverseV1($(\boldsymbol{y}, \boldsymbol{y}_a), (\bar{\boldsymbol{y}}, \bar{\boldsymbol{y}}_a)$)
  $\boldsymbol{z} \leftarrow \boldsymbol{y}_a$
  $\boldsymbol{x} \leftarrow \boldsymbol{y} - \boldsymbol{z}$, detach $\boldsymbol{x}$ from compute graph
  $\boldsymbol{x}_a \leftarrow \boldsymbol{z} - F(\boldsymbol{x})$
  $\bar{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{y}}_a + \bar{\boldsymbol{y}}$
  $\bar{\boldsymbol{x}} \leftarrow \bar{\boldsymbol{y}} + \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \bar{\boldsymbol{z}}$
  $\bar{\boldsymbol{x}}_a \leftarrow \bar{\boldsymbol{z}}$
  $\bar{\boldsymbol{w}} \leftarrow \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{w}}\right)^T \bar{\boldsymbol{z}}$
  detach $\boldsymbol{x}$ and $\boldsymbol{x}_a$ from compute graph
  return $(\boldsymbol{x}, \boldsymbol{x}_a)$, $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}_a)$ and $\bar{\boldsymbol{w}}$
end function

Algorithm 2 Simplified BlockReverse v2

function BlockReverseV2($(\boldsymbol{y}, \boldsymbol{y}_a), (\bar{\boldsymbol{y}}, \bar{\boldsymbol{y}}_a)$)
  $\boldsymbol{z} \leftarrow \boldsymbol{y}_a$, detach $\boldsymbol{z}$ from compute graph
  $\boldsymbol{x} \leftarrow \boldsymbol{y} - G(\boldsymbol{z})$
  $\boldsymbol{x}_a \leftarrow \boldsymbol{y}_a - \boldsymbol{x}$
  $\bar{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{y}}_a + \left(\frac{\partial G(\boldsymbol{z})}{\partial \boldsymbol{z}}\right)^T \bar{\boldsymbol{y}}$
  $\bar{\boldsymbol{x}} \leftarrow \bar{\boldsymbol{y}} + \bar{\boldsymbol{z}}$
  $\bar{\boldsymbol{x}}_a \leftarrow \bar{\boldsymbol{z}}$
  $\bar{\boldsymbol{w}} \leftarrow \left(\frac{\partial G(\boldsymbol{z})}{\partial \boldsymbol{w}}\right)^T \bar{\boldsymbol{y}}$
  detach $\boldsymbol{x}$ and $\boldsymbol{x}_a$ from compute graph
  return $(\boldsymbol{x}, \boldsymbol{x}_a)$, $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}_a)$ and $\bar{\boldsymbol{w}}$
end function

We now examine the input gradient at one layer in SRevNet v1, which is given by

$$\bar{\boldsymbol{x}} = \bar{\boldsymbol{y}} + \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \bar{\boldsymbol{y}} + \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \bar{\boldsymbol{y}}_a, \tag{16}$$

and compare it with a ResNet layer $\bar{\boldsymbol{x}} = \bar{\boldsymbol{y}} + \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \bar{\boldsymbol{y}}$; the only difference is that we add a third term $\left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \bar{\boldsymbol{y}}_a$ to ResNet. The input gradient in SRevNet v2 is defined as

$$\bar{\boldsymbol{x}} = \bar{\boldsymbol{y}} + \left(\frac{\partial G(\boldsymbol{y}_a)}{\partial \boldsymbol{y}_a}\right)^T \bar{\boldsymbol{y}} + \bar{\boldsymbol{y}}_a, \tag{17}$$

which means we simply replace $\boldsymbol{x}$ in ResNet with $\boldsymbol{y}_a = \boldsymbol{x} + \boldsymbol{x}_a$ and add a third term $\bar{\boldsymbol{y}}_a$. For RevNet, the input gradient is obtained by first computing the gradient for the first split $\boldsymbol{x}_1$:

$$\bar{\boldsymbol{x}}_1 = \bar{\boldsymbol{y}}_1 + \left(\frac{\partial G(\boldsymbol{y}_1)}{\partial \boldsymbol{y}_1}\right)^T \bar{\boldsymbol{y}}_2, \tag{18}$$

and then for the second split $\boldsymbol{x}_2$:

$$\bar{\boldsymbol{x}}_2 = \bar{\boldsymbol{y}}_2 + \left(\frac{\partial F(\boldsymbol{x}_2)}{\partial \boldsymbol{x}_2}\right)^T \bar{\boldsymbol{z}} = \bar{\boldsymbol{y}}_2 + \left(\frac{\partial F(\boldsymbol{x}_2)}{\partial \boldsymbol{x}_2}\right)^T \left(\bar{\boldsymbol{y}}_1 + \left(\frac{\partial G(\boldsymbol{y}_1)}{\partial \boldsymbol{y}_1}\right)^T \bar{\boldsymbol{y}}_2\right), \tag{19}$$

where $\bar{\boldsymbol{z}}$ stores an intermediate computation that is effectively equivalent to $\bar{\boldsymbol{x}}_1$, so that the full input gradient becomes $\bar{\boldsymbol{x}} = \mathrm{Concat}([\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2])$. Thus, $\bar{\boldsymbol{x}}_1$ and $\bar{\boldsymbol{x}}_2$ are computed with two different formulas, which is somewhat unintuitive given that they were split from the same input $\boldsymbol{x}$ and are afterward concatenated back as if they were one input gradient. For example, this means that the value of $\bar{\boldsymbol{x}}_1$ is propagated through $\bar{\boldsymbol{x}}_2$ and that we are essentially training two separate ResNet layer networks instead of one. Here, $\boldsymbol{x}_1$ serves mainly as an intermediate step for computing $\boldsymbol{x}_2$ and thus it would be more beneficial to dedicate an auxiliary variable for this purpose.

SRevNet trains somewhat more slowly than ResNet because activations must be recomputed. RevNet estimated this overhead by assuming a ResNet backward pass costs about $2\times$ the forward pass in multiply-adds, yielding an additional cost for SRevNet of approximately $(4-3)/3 = 1/3 \approx 33\%$ compute. This is a limitation of SRevNet, but the slowdown is exactly the same as with activation checkpointing or RevNet, since the two RevNet networks process equally sized input splits and together cost about as much as one ResNet-layer network.

3.2 Analysis of SRevNet v1 versus SRevNet v2

Although the distinction between SRevNet v1 ($G = I$) and v2 ($F = I$) seems subtle, it plays a key role in learning dynamics. The forward pass in SRevNet v1 at layer t, computed recursively, is

$$\boldsymbol{x}_a^{(t+1)} = \boldsymbol{x}_a^{(t)} + F(\boldsymbol{x}^{(t)}), \quad \boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \boldsymbol{x}_a^{(t+1)}. \tag{20}$$

In SRevNet v2, the forward pass is

$$\boldsymbol{x}_a^{(t+1)} = \boldsymbol{x}_a^{(t)} + \boldsymbol{x}^{(t)}, \quad \boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + G(\boldsymbol{x}_a^{(t+1)}). \tag{21}$$

Since $\boldsymbol{x}_a^{(1)}$ is initialized to zero, SRevNet v1 gives $\boldsymbol{x}_a^{(t+1)} = \sum_{i=1}^{t} F(\boldsymbol{x}^{(i)})$, while SRevNet v2 gives $\boldsymbol{x}_a^{(t+1)} = \sum_{i=1}^{t} \boldsymbol{x}^{(i)}$. In SRevNet v2, $\boldsymbol{x}_a^{(t+1)} = \sum_{i=1}^{t} \boldsymbol{x}^{(i)}$ naturally serves as a cumulative context buffer, whereas in SRevNet v1 it aggregates residuals $F(\boldsymbol{x}^{(i)})$, which lack direct semantic correspondence to input features. We first consider SRevNet v2 gradients, computed recursively as

$$\bar{\boldsymbol{x}}_a^{(t)} = \bar{\boldsymbol{x}}_a^{(t+1)} + \big(J_G^{(t+1)}\big)^T \bar{\boldsymbol{x}}^{(t+1)}, \quad \bar{\boldsymbol{x}}^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + \big(J_G^{(t+1)}\big)^T \bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{x}}_a^{(t+1)}, \tag{22}$$

where we set $J_G^{(t+1)} = \partial G(\boldsymbol{x}_a^{(t+1)})/\partial \boldsymbol{x}_a^{(t+1)}$ to simplify notation. Because the classifier uses only the output from the last layer L, $\boldsymbol{x}^{(L+1)}$, we have $\bar{\boldsymbol{x}}_a^{(L+1)} = 0$, so we can now rewrite $\bar{\boldsymbol{x}}_a^{(t)} = \sum_{i=t}^{L} \big(J_G^{(i+1)}\big)^T \bar{\boldsymbol{x}}^{(i+1)}$. The input gradient thus becomes

$$\begin{aligned} \bar{\boldsymbol{x}}^{(t)} &= \bar{\boldsymbol{x}}^{(t+1)} + \big(J_G^{(t+1)}\big)^T \bar{\boldsymbol{x}}^{(t+1)} + \sum_{i=t+1}^{L} \big(J_G^{(i+1)}\big)^T \bar{\boldsymbol{x}}^{(i+1)} \\ &= \bar{\boldsymbol{x}}^{(t+1)} + \sum_{i=t}^{L} \big(J_G^{(i+1)}\big)^T \bar{\boldsymbol{x}}^{(i+1)}. \end{aligned} \tag{23}$$

Compared with a standard ResNet $\bar{\boldsymbol{x}}^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + \big(J_F^{(t+1)}\big)^T \bar{\boldsymbol{x}}^{(t+1)}$, SRevNet v2 achieves the ResNet effect by providing a direct highway path for the upstream gradient from one layer above $t+1$, and it also creates direct multi-layer gradient pathways (a parallel ultra-highway effect) through which each layer receives direct gradient paths from all deeper layers. The SRevNet v1 gradients are

$$\bar{\boldsymbol{x}}_a^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{x}}_a^{(t+1)}, \quad \bar{\boldsymbol{x}}^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + \big(J_F^{(t+1)}\big)^T \big(\bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{x}}_a^{(t+1)}\big), \tag{24}$$

where $J_F^{(t+1)} = \partial F(\boldsymbol{x}^{(t)})/\partial \boldsymbol{x}^{(t)}$. We rewrite $\bar{\boldsymbol{x}}_a^{(t)} = \sum_{i=t}^{L} \bar{\boldsymbol{x}}^{(i+1)}$ and write the input gradient as

$$\begin{aligned} \bar{\boldsymbol{x}}^{(t)} &= \bar{\boldsymbol{x}}^{(t+1)} + \big(J_F^{(t+1)}\big)^T \big(\bar{\boldsymbol{x}}^{(t+1)} + \sum_{i=t+1}^{L} \bar{\boldsymbol{x}}^{(i+1)}\big) \\ &= \bar{\boldsymbol{x}}^{(t+1)} + \big(J_F^{(t+1)}\big)^T \big(\sum_{i=t}^{L} \bar{\boldsymbol{x}}^{(i+1)}\big). \end{aligned} \tag{25}$$

Here, however, we do not obtain the same direct parallel ultra-highway effect as in SRevNet v2, because $J_F^{(t+1)}$ scales each $\bar{\boldsymbol{x}}^{(i+1)}$ inside the sum. SRevNet v2 is conceptually similar to the cumulative residual strategy in DenseNet [20], which concatenates features from all preceding layers $[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}]$ so that each layer computes

$$\boldsymbol{x}^{(t+1)} = F\big(\mathrm{Concat}([\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}])\big). \tag{26}$$

Because $[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}]$ is concatenated along the feature dimension, we have

$$\bar{\boldsymbol{x}}^{(t)} = \sum_{i=t}^{L} \left(\frac{\partial \boldsymbol{x}^{(i+1)}}{\partial \boldsymbol{x}^{(t)}}\right)^T \bar{\boldsymbol{x}}^{(i+1)} = \sum_{i=t}^{L} \big(J_x^{(i+1)}\big)^T \bar{\boldsymbol{x}}^{(i+1)}. \tag{27}$$

DenseNet was designed to enable dense gradient connectivity (a parallel highway effect), where each layer receives direct gradient paths from all deeper layers. DenseNet argued that in ResNet, the identity function and the output of F are combined by summation, which may impede information flow: ResNet's highway effect bridges only adjacent layers, whereas DenseNet connects every layer to all deeper ones. In this sense, $\boldsymbol{x}_a$ in SRevNet v2 can be viewed as a replacement for the concatenation $[\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}]$ in DenseNet, but achieved using $\boldsymbol{x}_a$ summation rather than concatenation, preserving $\mathcal{O}(1)$ memory per layer instead of $\mathcal{O}(t)$. While both SRevNet versions achieve the standard ResNet effect, SRevNet v2 gives the gradient at each layer t direct gradient paths from all deeper layers $i > t$, where each path is scaled by the corresponding layer's Jacobian $\big(J_G^{(i+1)}\big)^T$. This creates a dense connectivity pattern analogous to DenseNet. In contrast, SRevNet v1 scales the entire accumulated gradient sum by a single Jacobian $\big(J_F^{(t+1)}\big)^T$; we hypothesize that this asymmetry distorts signals from deeper layers and may create a representational bottleneck. Appendix C gives another example of the different behavior of v1 and v2.

3.3 Analysis of Simplified RevNet versus Momentum ResNet

The neural-ODE-inspired Momentum ResNet (MResNet) in Eq. (9) uses a similar branching $G = I$ as SRevNet v1 (i.e., with asymmetric Jacobian scaling rather than the dense gradient highway of SRevNet v2), but adds momentum $\gamma \in [0, 1]$. However, as discussed in Section 2.3, γ imposes ODE-specific constraints, such as Highway Network-like scaling complications [13] and reconstruction errors. The input gradients in MResNet, computed recursively, are

$$\bar{\boldsymbol{v}}^{(t)} = \gamma(\bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{v}}^{(t+1)}), \quad \bar{\boldsymbol{x}}^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{v}}^{(t+1)}), \tag{28}$$

where $J_F^{(t+1)} = \partial F(\boldsymbol{x}^{(t)})/\partial \boldsymbol{x}^{(t)}$. With $\bar{\boldsymbol{v}}^{(L+1)} = 0$, unrolling yields $\bar{\boldsymbol{v}}^{(t)} = \sum_{i=t}^{L} \gamma^{i-t+1}\bar{\boldsymbol{x}}^{(i+1)}$, so

$$\begin{aligned}\bar{\boldsymbol{x}}^{(t)} &= \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\bar{\boldsymbol{x}}^{(t+1)} + \sum_{i=t+1}^{L} \gamma^{i-t}\bar{\boldsymbol{x}}^{(i+1)}) \\ &= \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\sum_{i=t}^{L} \gamma^{i-t}\bar{\boldsymbol{x}}^{(i+1)}).\end{aligned} \tag{29}$$

Because $(1-\gamma)$ scales each local Jacobian $J^{(t+1)}$, unrolling this recursion over L layers causes the Jacobian path to be $\prod_t (1-\gamma)^{L-t} J^{(t)}$ instead of $\prod_t J^{(t)}$ as it should normally be in ResNet, SRevNet and a plain network. In addition, the weighted momentum sum $(\sum_{i=t}^{L} \gamma^{i-t}\bar{\boldsymbol{x}}^{(i+1)})$ is distributed across these terms, injecting extra mixed powers of γ and $(1-\gamma)$ (e.g. $\propto \prod_t \gamma^{L-t}(1-\gamma)^{L-t}J^{(t)}$). This produces a depth-dependent multiplicative scaling pattern closely resembling the one of Highway Networks discussed in Section 2.2. The following example illustrates this effect: at layer L, $\bar{\boldsymbol{x}}^{(L)} = J_F^{(L+1)T}\bar{\boldsymbol{x}}^{(L+1)}$, $\bar{\boldsymbol{x}}^{(L)} = \big(\boldsymbol{I} + J_F^{(L+1)T}\big)\bar{\boldsymbol{x}}^{(L+1)}$ and $\bar{\boldsymbol{x}}^{(L)} = \big(\gamma\boldsymbol{I} + (1-\gamma)J_F^{(L+1)T}\big)\bar{\boldsymbol{x}}^{(L+1)}$ are the gradients for plain, ResNet, and Highway Network layers, respectively. Unrolling to layer $L-1$:

$$\begin{aligned}\text{plain: } \bar{\boldsymbol{x}}^{(L-1)} &= J_F^{(L)T} J_F^{(L+1)T}\bar{\boldsymbol{x}}^{(L+1)}, \\ \text{ResNet: } \bar{\boldsymbol{x}}^{(L-1)} &= \big(\boldsymbol{I} + J_F^{(L)T} + J_F^{(L+1)T} + J_F^{(L)T}J_F^{(L+1)T}\big)\bar{\boldsymbol{x}}^{(L+1)}, \\ \text{Highway: } \bar{\boldsymbol{x}}^{(L-1)} &= \big(\gamma^2\boldsymbol{I} + \gamma(1-\gamma)J_F^{(L)T} + \gamma(1-\gamma)J_F^{(L+1)T} + (1-\gamma)^2 J_F^{(L)T}J_F^{(L+1)T}\big)\bar{\boldsymbol{x}}^{(L+1)}.\end{aligned}$$

Highway Network thus exhibits the same problematic γ-scaling pattern as MResNet. The ResNet follow-up [14] showed that, although momentum is beneficial in SGD, which is updated once per step, in standard backprop the repeated γ scaling forms a long, undesirable multiplication series that harms training, e.g., via exploding/vanishing gradients along the Jacobian or highway path. The key to ResNet's success was to retain the plain network formulation and add purely additive terms for direct highway paths, keeping the paths direct rather than γ-downscaled with depth, so that information flows unimpeded through the layers—essential to combat the degradation problem. SRevNet aligns with this principle: rather than replacing the ResNet or plain network formulas, it only appends additive terms, so the forward/backward dynamics decompose into purely additive terms, which we believe makes it better for general backprop-based training. MResNet, by contrast, does not simply append additive terms; it significantly rescales and distorts the gradient flow, reducing the network's ability to learn the flow naturally and, in a sense, imposing a fixed inductive bias. The γ term is a by-product of MResNet's ODE formulation, but neural networks are discrete systems trained by backprop, and ODE-specific constraints do not directly apply to, e.g., a ResNet trained on ImageNet or a Transformer on natural language, where imposing γ can only introduce unnecessary complexity. We hypothesize that this is why MResNet fails more often than SRevNet for certain γ values, making the SRevNet vs MResNet debate parallel to the ResNet vs Highway discussion in Sections 2–3 of [14]. Appendix D extends the MResNet analysis. The γ factor also complicates exact reversible activations, since repeatedly multiplying/dividing by γ causes finite-precision loss; MResNet reversal requires $\boldsymbol{v}^{(t)} = \big(\boldsymbol{v}^{(t+1)} - (1-\gamma)F(\boldsymbol{x}^{(t)})\big)/\gamma$, which loses more recursively than γ-free SRevNet/RevNet. To combat this, MResNet used an algorithm from [21] that stores an additional information buffer of lost bits, but the main downside is this buffer must be stored in high 64-bit integer precision, consuming significant memory relative to SRevNet/RevNet in automatic mixed-precision float16 training.

4 Experiments

We benchmarked SRevNet on two architectures, the original ResNet based on convolutional networks [5] and the Transformer [15], to assess its generalizability across architectures. We

trained each model from scratch on public image and text datasets and compared SRevNet with standard ResNet/Transformer, RevNet, and MResNet. We coded each method as a custom `torch.autograd.Function` in PyTorch [22] with its own forward and backward definition. All models were built using the same PyTorch network class and training loop; the only difference was a conditional branch in the constructor that selected a specific `torch.autograd.Function`. We verified this by measuring FLOPs and including the network parameter count in the results, to ensure a fair comparison. The sole exception is RevNet, where we naturally must make changes on the architecture, but we did our best to match the depth and FLOPs by using the exact settings provided in the RevNet paper. The code needed to reproduce the results is provided in the supplementary material in the submission. Implementation details are given in Appendices A and B.

4.1 Image Experiments with Convolutional Neural Networks

Most experiments in RevNet were dedicated to image classification on ImageNet [23] with deep ResNets, so we found it necessary to benchmark on ImageNet. ImageNet is the gold standard in computer vision owing to its massive scale and diversity of high-quality annotated images. Table 1 shows that SRevNet roughly matches the accuracy of a deep 101-layer ResNet and is more parsimonious than MResNet, by matching the accuracy with a simpler design. The γ parameter makes MResNet effectively a set of distinct models rather than a single model, and our results demonstrate that SRevNet closely matches the best of these across seven different MResNet models. It achieves this by using a different branching structure while keeping a hypothesis space of size 1—a direct application of Occam's Razor. We also observed that MResNet fails for certain γ values; our theoretical analysis in Section 3.3 hypothesizes that this is due to the depth-dependent recursive multiplication by γ. We observed large gains in memory efficiency (Table 2) with $\sim$14 GiB reduction in memory usage compared to standard ResNet, where irreversible downsampling layers limited further reductions. The observed training-time overhead was approximately 30%, consistent with our theoretical estimates.

Table 1: Classification accuracy on ImageNet with ResNet101 architecture.

METHOD	VAL TOP-1 (ACC.)
RESNET	78.2
REVNET	76.7
MRESNET $\gamma = 0.1$	0.1
$-\ \gamma = 0.25$	0.1
$-\ \gamma = 0.5$	0.1
$-\ \gamma = 0.75$	77.2
$-\ \gamma = 0.9$	77.3
$-\ \gamma = 0.95$	76.9
$-\ \gamma = 0.99$	76.0
SREVNET V1	77.0
SREVNET V2	77.3

Table 2: Training efficiency metrics on ImageNet with ResNet101 architecture.

METHOD	MEMORY (GIB)	NETWORK (MIB)	FLOPS (TFLOPS)	TRAIN TIME (HOURS)
RESNET	32.4	42.5	23.9	20.3
REVNET	21.6	43.3	30.5	27.8
MRESNET	18.6	42.5	30.3	29.6
MRESNET γ-ALG.	33.2	42.5	30.3	-
SREVNET V1	18.3	42.5	30.3	26.5
SREVNET V2	18.3	42.5	30.3	26.4

As noted in Section 2.3, although zero activation storage is theoretically possible, practical limits exist due to irreversible layers; this applies to all reversible networks, not just SRevNet. This limits further reductions in memory consumption and also explains why training is somewhat faster: we apply the reversible procedure only to the reversible ResNet layers, skipping the initial *conv1* layer,

the last classifier network, and the handful of irreversible downsampling layers. Downsampling layers are key components of convolutional ResNets, yet they are troublesome in reversible architectures because a reversible function must be one-to-one, whereas operations such as downsampling create many-to-one mappings and are therefore irreversible. Overcoming this would require modifying ResNet itself, for example via i-RevNet [24], but such major architectural overhauls defeat the purpose of this paper. Here, we retained the original downsampling layers, since otherwise the element-wise additions in Eq. (10)–(13) would encounter dimension mismatches. While we must still store activations for running backprop within a single ResNet layer F block, we can reuse that memory space across other layers, unlike standard backprop. MResNet's γ also complicates exact reversion in finite-precision training, unlike the γ-free SRevNet and RevNet: as noted in Section 3.3, its exact γ inversion algorithm requires a 64-bit buffer, which under our float16 training would increase memory to 33.2 GiB—nearly the same as ResNet's. Since our focus is practical memory efficiency, we decided not to use the γ algorithm in our experiments.

4.2 Language Modeling Experiments with Transformers

Transformers are now used in nearly every large language model (LLM). More generally, a Transformer can be viewed as a ResNet whose layers alternate between self-attention and feed-forward networks. Unlike ResNets, Transformers have no downsampling layers, creating opportunities for further memory savings. We trained all methods on a Llama [25–27] 300M model using the LitGPT [28] library. Llama is an open-source state-of-the-art decoder-only Transformer architecture like GPT [29–32] that excels at language modeling and text generation. We evaluated performance using validation perplexity on the OpenWebText dataset [33]. Overall, our findings were similar to those for ResNet on ImageNet. We observed that SRevNet's perplexity closely matched that of standard Llama (Table 3), and that SRevNet can match MResNet's perplexity while offering better robustness by removing its dependence on the momentum term. Our experiments on ImageNet and OpenWebText (and preliminary results on CIFAR) confirm that SRevNet v2 consistently outperforms SRevNet v1, which we believe is due to $F = I$ branching being more promising than $G = I$, as detailed in Section 3.2. Table 4 shows that SRevNet reduces GPU memory usage substantially compared to standard Llama, achieving a $\sim$48 GiB reduction while keeping the training-time overhead at $\sim$30%. We applied the reversible procedure only to the core Transformer layers, i.e., those after the initial embedding layer and before the final classifier network, which can be computationally expensive for large vocabularies. As in the ResNet experiments, we must still use standard backprop within the F block of a single layer, since the softmax in attention and the activation functions in the feed-forward networks prevent complete reversibility; however, SRevNet can reuse the memory space needed for one layer in other layers. The MResNet models that failed on ImageNet also failed here: as shown in Tables 1 and 3, MResNet consistently fails for certain γ values across two different architectures and two data domains. We therefore take this as strong evidence that the failure is intrinsic to the γ-dependent formulation. This parallels the comparison of ResNet with Highway Network in Figure 1 of the ResNet follow-up [14], where Highway Network fails for many γ variants whereas ResNet achieves similar accuracy with a much simpler design.

Table 3: Perplexity on OpenWebText with Llama model architecture.

METHOD	VAL PPL (VAL.)
LLAMA	16.4
REVNET	16.5
MRESNET $\gamma = 0.1$	921.3
$-\ \gamma = 0.25$	168.9
$-\ \gamma = 0.5$	128.4
$-\ \gamma = 0.75$	51.5
$-\ \gamma = 0.9$	16.8
$-\ \gamma = 0.95$	17.0
$-\ \gamma = 0.99$	18.3
SREVNET V1	17.5
SREVNET V2	16.8

Table 4: Key training efficiency metrics from the Llama runs on OpenWebText.

METHOD	MEMORY (GIB)	NETWORK (GIB)	FLOPS (TFLOPS)	TRAIN TIME (HOURS)
LLAMA	64.8	1.1	63.3	17.2
REVNET	17.4	1.3	82.3	21.8
MRESNET	17.0	1.1	82.3	22.7
SREVNET V1	17.0	1.1	82.3	22.3
SREVNET V2	17.0	1.1	82.3	22.2

4.3 Comparison with Activation Checkpointing

In Table 5, we validate that as the number of layers increases, the memory usage of SRevNet remains constant, unlike activation checkpointing (AC), for which memory usage still scales linearly with the number of layers. This means that SRevNet incurs the same 33% training-time overhead as AC but makes memory usage independent of network depth, which is a significant advantage. In Table 6, we verify that the forward- and backward-pass FLOP counts match those of AC, while also confirming the 33% increase in FLOPs during the backward pass compared to standard ResNet.

Table 5: GPU memory usage from the combined forward and backward passes at different depths.

LAYERS	100	200	300	400
RESNET	4.9 GIB	9.7 GIB	14.5 GIB	19.3 GIB
RESNET AC	2.6 GIB	5.0 GIB	7.4 GIB	9.8 GIB
SREVNET V1	0.3 GIB	0.3 GIB	0.3 GIB	0.3 GIB
SREVNET V2	0.3 GIB	0.3 GIB	0.3 GIB	0.3 GIB
NETWORK	0.9 MIB	1.8 MIB	2.7 MIB	3.6 MIB

Table 6: Backward pass (first number) and forward pass (second number) GFLOPs at various depths.

LAYERS	100	200	300	400
RESNET	739.9 (369.9)	1479.8 (739.9)	2219.6 (1109.8)	2959.5 (1479.8)
RESNET AC	1109.8 (369.9)	2219.6 (739.9)	3329.4 (1109.8)	4439.3 (1479.8)
SREVNET V1	1109.8 (369.9)	2219.6 (739.9)	3329.4 (1109.8)	4439.3 (1479.8)
SREVNET V2	1109.8 (369.9)	2219.6 (739.9)	3329.4 (1109.8)	4439.3 (1479.8)

5 Conclusion and Future Work

This work introduced Simplified RevNet (SRevNet), a new formulation of RevNet that can serve as a drop-in replacement for existing ResNet architectures. Setting one of the functions F or G in RevNet to the identity function naturally yields two variants of SRevNet. We find that the two variants differ meaningfully: SRevNet v1 accumulates residual outputs, whereas SRevNet v2 accumulates input features and behaves like a cumulative context buffer, producing a DenseNet-like parallel gradient highway, which makes SRevNet v2 a more direct generalization of ResNet than SRevNet v1. We showed that SRevNet performs comparably to similarly sized ResNets, but with significant improvements in memory efficiency. Compared to prior work on reversible networks, SRevNet is simpler and more practical, and thus we believe that it offers stronger generality and potential for broader adoption. While most software libraries now offer activation checkpointing via a command-line argument, making the network reversible has been challenging because of the architectural modifications required and has not yet reached mainstream adoption. However, we believe that enabling reversibility via a command-line argument should be just as simple, as it is a natural extension of activation checkpointing once RevNet's constraints are removed, which improves memory efficiency without affecting computational cost. As future work, we are currently working on adapting SRevNet to IO-aware attention algorithms to enable reversible linear self-attention by using a recurrent formulation that can leverage parallel processing to achieve faster runtimes.

Acknowledgments

This work was supported by the Department of Computer Science, Norwegian University of Science and Technology (NTNU).

References

[1] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *Deep Learning*, pages 399–467. Springer International Publishing, Cham, 2023. ISBN 978-3-031-38747-0. doi: 10.1007/978-3-031-38747-0_10. URL https://doi.org/10.1007/978-3-031-38747-0_10.

[2] Christopher M Bishop and Hugh Bishop. *Deep learning: Foundations and concepts*. Springer Nature, 2023.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[7] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf.

[8] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1627–1635. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/jaderberg17a.html.

[9] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4839–4850. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/nokland19a.html.

[10] Robert Edwin Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.

[11] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

[12] Louis Fournier, Stéphane Rivaud, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Can forward gradient match backpropagation? In *International Conference on Machine Learning*, pages 10249–10264. PMLR, 2023.

[13] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[16] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[17] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[18] Michael E Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. Momentum residual neural networks. In *International Conference on Machine Learning*, pages 9276–9287. PMLR, 2021.

[19] Christian Lubich and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006.

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2261–2269. Ieee, 2017.

[21] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2113–2122, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/maclaurin15.html.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[24] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

[25] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[27] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[28] Lightning AI. Litgpt. https://github.com/Lightning-AI/litgpt, 2023. Commit SHA: df94ee2cae47d8d2fb20815c175e98fd1f64a27a.

[29] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[31] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[32] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[33] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

[34] PyTorch. Imagenet training in pytorch. `https://github.com/pytorch/examples/tree/main/imagenet`, 2016. Commit SHA: 42068585f7da9a2f8656f793d21c30ee6806c6b9.

[35] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[37] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[38] Zixiao Ken Wang. Microllama: A 300m-parameter language model trained from scratch. `https://github.com/keeeeenw/MicroLlama`, `https://huggingface.co/keeeeenw/MicroLlama`, 2024. GitHub and Hugging Face repositories.

[39] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

[40] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

A Implementation Details Image Experiments with Convolutional Neural Networks

In the image experiments, we trained on the ImageNet-1K 2012 dataset [23] for image classification. ImageNet-1K consists of 1.28 million training images and 50,000 validation images spanning 1,000 classes. The dataset was downloaded from the official ImageNet website. We trained each model on the training images and measured the performance based on the top-1 accuracy (measured in percentage) on the validation images. We downloaded the most recent version of PyTorch's ImageNet training code from the official PyTorch GitHub repository [34], and most of the training hyperparameters were directly taken from that training script. The only changes we made to the training script were to enable automatic mixed-precision (AMP) training with float16 precision to speed up training and adopt a cosine-annealing learning-rate scheduler with linear warmup. We used AMP that was available in native PyTorch. We used batch size $= 512$, distributed across two NVIDIA RTX4090 24GB VRAM GPUs (256 images per GPU) with multi-processing Distributed Data Parallel training. We trained each method using the standard Stochastic Gradient Descent (SGD) optimizer with base learning rate $= 0.3$, momentum $= 0.9$, and weight decay $= 0.0001$ as the L2 penalty coefficient, for 90 epochs. The learning rate was linearly warmed up from 0 to 0.3 over the first 10 epochs. After warmup, the learning rate was decayed by a cosine learning rate schedule [35] until it reached a factor 0.001 of its base value (i.e., 0.3×0.001). The learning rate was updated at the end of each training step (i.e., after each train batch). All models were optimized using standard cross-entropy loss. All images underwent standard preprocessing, using the procedure and settings described in the Inception [36] paper. The training images were randomly cropped and resized to 224×224 resolution, followed by random horizontal flipping, and then each value was

rescaled to be within 0 and 1. The validation images were resized to 256×256 resolution, then center cropped to 224×224, and re-scaled so that each value was within 0 and 1. We used the image augmentations that were already in the PyTorch ImageNet training code, which were implemented in the Torchvision library [37]. All models were based on the ResNet101 [5] architecture that was implemented in Torchvision. To create an equivalent RevNet, we adopted the model configuration described in the original RevNet [7] paper. The main model configurations are shown in Table 7. The PyTorch ImageNet training code on GitHub is licensed under a permissive BSD 3-Clause License.

CONFIGURATION	RESNET	REVNET
NUMBER OF CHANNELS	64-128-256-512	128-256-512-832
NUMBER OF LAYERS	3-4-23-3	2-2-11-2
BOTTLENECK	✓	✓

Table 7: Major model configurations in the ResNet and RevNet models used on ImageNet. Here we assume that a ResNet is organized into four main stages of residual layers after the initial *conv1*-layer, and the fields "Number of channels" and "Number of layers" denote the value in each stage. The "Bottleneck" field indicates whether the residual block uses the *Bottleneck* or *Basic* variant.

B Language Modeling Experiments with Transformers

For the language modeling experiments, we downloaded the source code from the LitGPT GitHub repository [28] and integrated our reversible methods into their pretraining recipes. We trained each method using one of their Llama [25–27] variants with 300M parameters, which is known as MicroLlama [38]. We adopted the TinyLlama [39] tokenizer, which adopted the same tokenizer as Llama2 [26]. We retained LitGPT's model architecture; the main model configurations are summarized in Table 8. We used the pre-made training configurations for Llama-300M in LitGPT but set macro batch size $= 512$ (instead of the default 48) and micro batch size $= 16$ (instead of the default 12). We trained with gradient accumulation and did a gradient descent step after 32 training steps. All models were trained using the AdamW optimizer [40]. We used the following default settings: learning rate $= 0.0004$, weight decay $= 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.95$. The learning rate was warmed up during the first 2000 training steps and annealed with a cosine scheduler [35] towards a minimum learning rate of 0.00005. We encountered some training instability with the default settings for our first version, Simplified RevNet v1, so we lowered the learning rate to 0.00024 and the weight decay to 0.01 for this method only. We trained all methods in full precision on one NVIDIA H100 80GB VRAM GPU on a Slurm cluster, on a total of 4B tokens. We used LitGPT's default value val_split_fraction $= 0.0005$ as the fraction of data that should be put aside for validation. We downloaded and processed the OpenWebText dataset [33] through the provided scripts in LitGPT. The tokenizer was also downloaded through LitGPT. Because some care is required using custom `torch.autograd.Functions` with `torch.compile` and fused optimizers, we decided to disable `torch.compile` and fused optimizers. Otherwise, we made no changes to the LitGPT source code. To create an equivalent RevNet, we tried to match the depth and number of parameters with standard Llama. Table 9 summarizes the model configurations for RevNet. The LitGPT GitHub repository is licensed under a permissive Apache License 2.0.

Configuration	Value
Hidden dimension	1024
Attention heads	16
Block size	2048
Vocab size	32000
Padding multiple	64
Layers	12
Rotary percentage	1.0
Bias	False
Norm	RMSNorm
Norm epsilon	1×10^{-5}
Intermediate size	5632
Query groups	4

Table 8: Model configurations for Llama taken from LitGPT.

Configuration	Value
Hidden dimension	2048 (1024 per branch)
Attention heads	16
Block size	2048
Vocab size	32000
Padding multiple	64
Layers	12 (6 per branch)
Rotary percentage	1.0
Bias	False
Norm	RMSNorm
Norm epsilon	1×10^{-5}
Intermediate size	5632
Query groups	4

Table 9: RevNet model configurations implemented in LitGPT.

C Additional Example of Different Behavior Between SRevNet v1 and SRevNet v2

Another factor that may explain the different behavior of SRevNet v1 and SRevNet v2 is that the functional asymmetry noted in Section 3.2 is also visible in the forward pass. In SRevNet v1, transformation precedes accumulation:

$$\begin{aligned} \boldsymbol{x}_a^{(t+1)} &= \boldsymbol{x}_a^{(t)} + F(\boldsymbol{x}^{(t)}) \quad \text{(transform)}, \\ \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} + \boldsymbol{x}_a^{(t+1)} \quad \text{(accumulate)}. \end{aligned} \tag{30}$$

In SRevNet v2, by contrast, the accumulation is performed first and the transformation second:

$$\begin{aligned} \boldsymbol{x}_a^{(t+1)} &= \boldsymbol{x}_a^{(t)} + \boldsymbol{x}^{(t)} \quad \text{(accumulate)}, \\ \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} + G(\boldsymbol{x}_a^{(t+1)}) \quad \text{(transform)}. \end{aligned} \tag{31}$$

The order of the transformation and accumulation steps affects the input to the network. If the network input is decomposed into a single $\boldsymbol{x}$ term and a single $\boldsymbol{x}_a$ term, then in SRevNet v1 the transformation is applied to $F(\boldsymbol{x}^{(t)}) = F(\boldsymbol{x}^{(t-1)} + \boldsymbol{x}_a^{(t)})$, which represents an asymmetric decomposition using $\boldsymbol{x}$ from the previous layer and $\boldsymbol{x}_a$ from the current layer. In SRevNet v2, by contrast, the transformation is applied to $G(\boldsymbol{x}_a^{(t+1)}) = G(\boldsymbol{x}^{(t)} + \boldsymbol{x}_a^{(t)})$, making the decomposition symmetric because both $\boldsymbol{x}$ and $\boldsymbol{x}_a$ are taken from the current layer. In other words, this distinction is not simply about t versus $t-1$, but about whether the network function has access to a joint state $(\boldsymbol{x}_a^{(t)}, \boldsymbol{x}^{(t)})$ or only to a marginal, single-branch state $\boldsymbol{x}^{(t)}$. This suggests that SRevNet v1 may operate on disjoint information sets, creating a representational bottleneck.

D Extended Analysis of Simplified RevNet versus Momentum ResNet

The input gradients in MResNet, computed recursively, are

$$\bar{\boldsymbol{v}}^{(t)} = \gamma(\bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{v}}^{(t+1)}), \quad \bar{\boldsymbol{x}}^{(t)} = \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\bar{\boldsymbol{x}}^{(t+1)} + \bar{\boldsymbol{v}}^{(t+1)}),$$

where we have set $J_F^{(t+1)} = \partial F(\boldsymbol{x}^{(t)})/\partial \boldsymbol{x}^{(t)}$. With $\bar{\boldsymbol{v}}^{(L+1)} = 0$, unrolling yields $\bar{\boldsymbol{v}}^{(t)} = \sum_{i=t}^{L} \gamma^{i-t+1}\bar{\boldsymbol{x}}^{(i+1)}$. The input gradient can therefore be written as

$$\begin{aligned}\bar{\boldsymbol{x}}^{(t)} &= \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\bar{\boldsymbol{x}}^{(t+1)} + \sum_{i=t+1}^{L} \gamma^{i-t}\bar{\boldsymbol{x}}^{(i+1)}) \\ &= \bar{\boldsymbol{x}}^{(t+1)} + (1-\gamma)\big(J_F^{(t+1)}\big)^T(\sum_{i=t}^{L} \gamma^{i-t}\bar{\boldsymbol{x}}^{(i+1)}).\end{aligned}$$

As discussed in Section 3.3, this recursion produces gradient terms with exponential multiplicative γ weighting and mixed $\gamma(1-\gamma)$, e.g. $\propto \prod_t (1-\gamma)^{L-t} J^{(t+1)}$ and mixed $\propto \prod_t \gamma^{L-t}(1-\gamma)^{L-t} J^{(t+1)}$ as opposed to the clean unscaled $\prod_t J^{(t+1)}$ as it should normally be in ResNet, SRevNet and a plain network. We now illustrate this with a basic example that captures the essence of this effect. The coupling between $\boldsymbol{x}$ and $\boldsymbol{v}$ makes the recursion tedious, however, we can unroll it from the final layer with $\bar{\boldsymbol{x}}^{(L+1)}$ as the incoming loss gradient. At layer L:

$$\bar{\boldsymbol{x}}^{(L)} = \bar{\boldsymbol{x}}^{(L+1)} + (1-\gamma)\big(J^{(L+1)}\big)^T\bar{\boldsymbol{x}}^{(L+1)} + (1-\gamma)\big(J^{(L+1)}\big)^T\gamma\bar{\boldsymbol{x}}^{(L+1)}.$$

Layer $L-1$:

$$\bar{\boldsymbol{x}}^{(L-1)} = \bar{\boldsymbol{x}}^{(L)} + (1-\gamma)\big(J^{(L)}\big)^T\bar{\boldsymbol{x}}^{(L)} + (1-\gamma)\big(J^{(L)}\big)^T(\gamma\bar{\boldsymbol{x}}^{(L)} + \gamma^2\bar{\boldsymbol{x}}^{(L+1)}),$$

which we decompose based on each three $\bar{\boldsymbol{x}}^{(L)}$ terms above

$$\bar{\boldsymbol{x}}^{(L-1)} = G_1 + G_2 + G_3,$$

where

$$G_1 = \bar{\boldsymbol{x}}^{(L)}$$

is the upstream gradient,

$$G_2 = (1-\gamma)\left(J^{(L)}\right)^T\bar{\boldsymbol{x}}^{(L+1)} + (1-\gamma)(1-\gamma)\left(J^{(L)}\right)^T\left(\big(J^{(L+1)}\big)^T(\bar{\boldsymbol{x}}^{(L+1)} + \gamma\bar{\boldsymbol{x}}^{(L+1)})\right)$$

$$\begin{aligned}G_3 = \gamma\Bigg((1-\gamma)\left(J^{(L)}\right)^T\bar{\boldsymbol{x}}^{(L+1)} + (1-\gamma)(1-\gamma)\left(J^{(L)}\right)^T\left(\big(J^{(L+1)}\big)^T(\bar{\boldsymbol{x}}^{(L+1)} + \gamma\bar{\boldsymbol{x}}^{(L+1)})\right)\Bigg) \\ +\gamma^2(1-\gamma)\left(J^{(L)}\right)^T\bar{\boldsymbol{x}}^{(L+1)},\end{aligned}$$

or more concisely

$$G_3 = \gamma G_2 + \gamma^2(1-\gamma)\left(J^{(L)}\right)^T\bar{\boldsymbol{x}}^{(L+1)}.$$

Now observe that the terms that connect Jacobians $J^{(L)}J^{(L+1)}$ in G_2 and G_3 are scaled by $(1-\gamma)^2$. This pattern will repeat and create nasty exponential scaling patterns, e.g., $\propto (1-\gamma)^L$ in G_2 and $\propto \gamma^L(1-\gamma)^L$ in G_3. We rewrite more generally for layer t:

$$G_1^{(t)} = G_1^{(t+1)} + G_2^{(t+1)} + G_3^{(t+1)} = \bar{\boldsymbol{x}}^{(t+1)}$$

$$G_2^{(t)} = (1-\gamma)\left(J^{(t+1)}\right)^T(G_1^{(t+1)} + G_2^{(t+1)} + G_3^{(t+1)})$$

$$G_3^{(t)} = \gamma G_2^{(t)} + (1-\gamma)\left(J^{(t+1)}\right)^T\sum_{j=t}^{L}\gamma^{j-t+1}(G_1^{(j+1)} + G_2^{(j+1)} + G_3^{(j+1)}).$$

Because only $G_2^{(t)}$ and $G_3^{(t)}$ contain the local Jacobian $J^{(t+1)}$ and each is scaled by $(1-\gamma)$, recursively applying this will cause the Jacobian path to be $\prod_t (1-\gamma)^{L-t} J^{(t+1)}$ instead of $\prod_t J^{(t+1)}$ as it should normally be in ResNet, SRevNet and a plain network. Additionally, $G_3^{(t)}$ also involves a sum with an exponential γ factor because of $\bar{\boldsymbol{v}}^{(t)} = \sum_{j=t}^{L} \gamma^{j-t+1} \bar{\boldsymbol{x}}^{(j+1)}$ and is distributed into the other terms, which makes it even nastier, e.g. $\propto \prod_t \gamma^{L-t}(1-\gamma)^{L-t} J^{(t+1)}$. This creates unnecessary complicated expressions that involve repeatedly scaling gradients by γ terms.

For comparison, the gradient in a ResNet at layer L is:

$$\bar{\boldsymbol{x}}^{(L)} = \left(\boldsymbol{I} + (J^{(L+1)})^T \right) \bar{\boldsymbol{x}}^{(L+1)},$$

and for layer $L-1$:

$$\bar{\boldsymbol{x}}^{(L-1)} = \left(\boldsymbol{I} + (J^{(L)})^T \right) \bar{\boldsymbol{x}}^{(L)} = \left(\boldsymbol{I} + (J^{(L)})^T + (J^{(L+1)})^T + (J^{(L)})^T (J^{(L+1)})^T \right) \bar{\boldsymbol{x}}^{(L+1)}.$$

The gradient in a plain neural network at layer L is:

$$\bar{\boldsymbol{x}}^{(L)} = (J^{(L+1)})^T \bar{\boldsymbol{x}}^{(L+1)},$$

and for layer $L-1$:

$$\bar{\boldsymbol{x}}^{(L-1)} = (J^{(L)})^T (J^{(L+1)})^T \bar{\boldsymbol{x}}^{(L+1)}.$$

In a Highway network the gradient at layer L is:

$$\bar{\boldsymbol{x}}^{(L)} = \left(\gamma \boldsymbol{I} + (1-\gamma)(J^{(L+1)})^T \right) \bar{\boldsymbol{x}}^{(L+1)},$$

and for layer $L-1$:

$$\bar{\boldsymbol{x}}^{(L-1)} = \left(\gamma \boldsymbol{I} + (1-\gamma)(J^{(L)})^T \right) \bar{\boldsymbol{x}}^{(L)} = \left(\gamma^2 \boldsymbol{I} + \gamma(1-\gamma)(J^{(L)})^T + \gamma(1-\gamma)(J^{(L+1)})^T + (1-\gamma)^2 (J^{(L)})^T (J^{(L+1)})^T \right) \bar{\boldsymbol{x}}^{(L+1)}.$$

Here we see a similar nasty γ scaling pattern as in MResNet.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction, about memory efficiency, training speed and architectural flexibility, are empirically verified in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The lower memory at the cost of longer training is theoretically estimated in the end of Section 3 and verified in Section 4 via measured runtimes and FLOPs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.

- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification:

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Forward and reverse steps are defined in Eq. (10)–(13), the backprop procedure is detailed in Algorithm 1 and 2. Architecture and training details are disclosed in Section 4 and Appendix A–B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.

- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We trained on publicly available datasets in all experiments. The source code and training scripts are available in the supplementary material in the submission.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Experimental details are specified in Section 4 and Appendix A-B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Due to high computational training costs, we conducted each experiment only once.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide information about the computer resources we used for training in Appendix A-B. We also report memory usage and runtime in Section 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms, in every aspect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper is foundational research. While reducing memory usage can make neural network training more accessible, it does not create a societal concern.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification:

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit original owners of assets in Section 4 and Appendix A–B. URLs to the external GitHub repositories and their exact commit hashes are provided in the References.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We coded our method as a custom subclass of `torch.autograd.Function` in PyTorch [22], following PyTorch's guidelines. The code is included in the supplementary material and we plan to make the code publicly available.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification:

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification:

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification:

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.