Data-free Neural Representation Compression with Riemannian Neural Dynamics

Zhengqi Pei [1 2] **Anran Zhang** [1 2] **Shuhui Wang** [1 3] **Xiangyang Ji** [4] **Qingming Huang** [5 1 3]

Abstract

Neural models are equivalent to dynamic systems from a physics-inspired view, implying that computation on neural networks can be interpreted as the dynamical interactions between neurons. However, existing work models neuronal interaction as a weight-based linear transformation, and the nonlinearity comes from the nonlinear activation functions, which leads to limited nonlinearity and data-fitting ability of the whole neural model. Inspired by Riemannian geometry, we interpret neural structures by projecting neurons onto the Riemannian neuronal state space and model neuronal interaction with Riemannian metric ($RieM$), which provides a neural representation framework with higher parameter efficiency. With $RieM$, we further design a novel data-free neural compression mechanism that does not require additional fine-tuning with real data. Using backbones like ResNet and Vision Transformer, we conduct extensive experiments on datasets such as MNIST, CIFAR-100, ImageNet-1k, and COCO object detection. Empirical results show that, under equal compression rates and computational complexity, models compressed with $RieM$ achieve superior inference accuracy compared to existing data-free compression methods.

1. Introduction

Integrating neural models with dynamical systems can provide theoretically solid insights into the learning theory of neural representation (Liu & Theodorou, 2019; Li et al., 2022a; Gu & Dao, 2023; Pei & Wang, 2023). Mathematically, any weight-based neural architecture can be interpreted as neurons in a Euclidean neuronal state space (Pei & Wang, 2023), which treats the neural weights as the dynamic relation between neuronal states in Euclidean neuronal state space. This dynamics-inspired approach transforms a weights-based neural model into a neurons-based one, which typically requires fewer parameters since a single neuron can be associated with many neural weights. Thus, neurons seem like a more global unit of representation. However, Euclidean space's overly simplified nature results in a lower and underfitted representation capacity for the neural models. It is necessary to interpret and analyze neural models using more flexible nonlinear metrics beyond those accompanied by Euclidean space.

Among various mathematical frameworks, Riemannian geometry (Petersen, 2006) extends the classical Euclidean spaces via the Riemannian manifolds dealing with curved nonlinear spaces (Lee & Lee, 2012). In the context of neural models, Riemannian geometry excels in depicting nonlinearity, offering greater flexibility in modeling the connections among neurons compared to Euclidean spaces. Riemannian metrics can capture interactions between different dimensions of the neuronal state space. They treat these interdimensional relations as additional hidden dimensions without adding extra trainable parameters. This feature is critical because we require a highly expressive metric function to define relations between neurons, enabling the representation of every weight matrix in neural models in a neuron-centric manner with fewer parameters. If we were to compute relations between neurons using Euclidean metrics, the resulting weight matrix would be a low-rank distance matrix. This would significantly diminish the expressive power in comparison to weights-based neural models, highlighting the limitations of Euclidean metrics in this context. Therefore, it is indispensable to introduce nonlinear Riemannian metrics to enhance the expressive power of neuron-based models. Theoretically, the resulting Riemannian neuronal state space can possess higher representational capacity and parameter efficiency.

There have been several attempts (Shao et al., 2018; Kaul & Lall, 2019) to interpret data modeling using Riemannian

[1]Institute of Computing Technology, Chinese Academy of Sciences. [2]School of Artificial Intelligence, University of Chinese Academy of Sciences. [3]Peng Cheng Laboratory. [4]Department of Automation, Tsinghua University. [5]School of Computer Science and Technology, University of Chinese Academy of Sciences.. Correspondence to: Shuhui Wang <wangshuhui@ict.ac.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

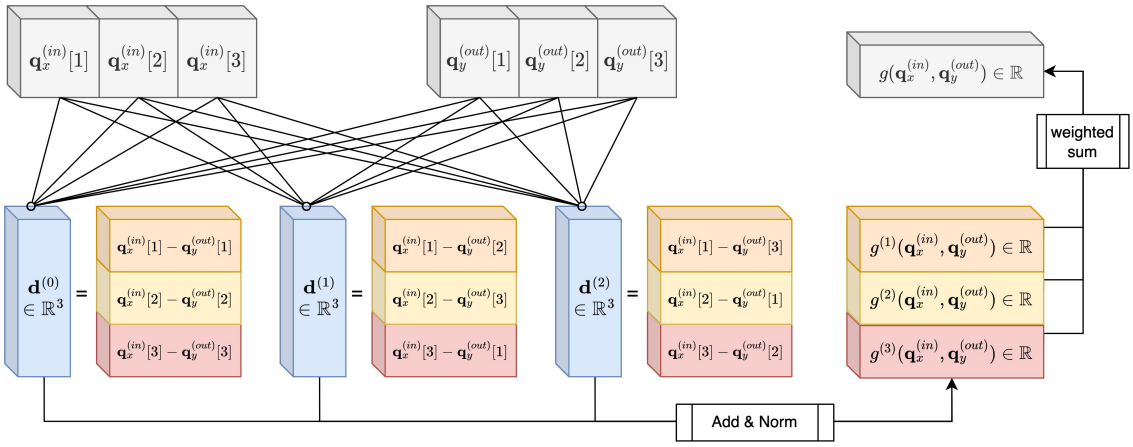

Figure 1: **Computing a Neural Riemannian Metric**: We interpret an arbitrary weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ of a neural model as the dynamical interactions between m input neurons and n output neurons. Each neuron is interpreted as D_q-dimensional dynamical state, *i.e.*, $\mathbf{q}_x^{(in)},\ \mathbf{q}_y^{(out)} \in \mathbb{R}^{D_q}$, with $x \in [1, m],\ y \in [1, n]$. We assume $D_q = 3$ in this figure. Following this, we interpret each weight value $\mathbf{W}_{xy}$ as the scalar-based relation measured by a Riemannian metric $g : \mathbb{R}^{D_q} \times \mathbb{R}^{D_q} \to \mathbb{R}$ such that $\mathbf{W}_{xy} = g(\mathbf{q}_x^{(in)}, \mathbf{q}_y^{(out)})$. The exact analytical form of this Riemannian metric is as follows. The dimensional components of $\mathbf{q}_x^{(in)}$ and $\mathbf{q}_y^{(out)}$ are mutually operated to generate D_q displacement vectors, *i.e.*, $\mathbf{d}^{(s)} \in \mathbb{R}^{D_q},\ s \in [1, D_q]$. Then, we merge these displacement vectors via a non-linear summation operation, followed by a $\mathbb{R}^{D_q} \mapsto \mathbb{R}$ projection that results in the scalar-based relation, which is expected to approximate each weight value.

geometry. However, the nonlinear neuronal interactions and the dynamic neuronal system have not been well studied from the Riemannian geometry perspective (Udriste & Tevy, 2020). To better integrate Riemannian dynamics with neural models, we propose a Riemannian metric tailored for neural models *RieM* to characterize interactions between neurons. Unlike Pei & Wang (2023), which interprets a neural weight linking two neurons as the path integral between the neural states using Euclidean metrics, we use Riemannian metrics instead without needing fine-tuning with real data. The only parameters in the storage are the dynamical states of neurons and the trainable parameters contained by Riemannian metrics. Recall that the primary advantage of Riemannian metrics is their ability to measure the relations between all the dimensional components of two embeddings without adding extra trainable parameters. They support comparisons across dimensions, unlike L_p-norms, which only facilitate comparisons within the same dimension. Thus, the Riemannian neural interactions can capture more intrinsic features comprehensively. To be compatible with mainstream parallel computing pipelines, we also design several complementary techniques, *e.g.*, displacement matrix, shared correlation counts, dynamical Merging, *etc.*, to better integrate Riemannian dynamics with neural models.

The experimental section focuses on how *RieM* facilitates more efficient data-free model compression and accelerates model inference. We apply *RieM*-based neural compression to various vision models, including CNN-based models, Transformer-based models, and the models from the DETR family. We conduct experiments on public datasets including MNIST (Deng, 2012), CIFAR-100 (Krizhevsky et al., 2009), ImageNet-1k (Deng et al., 2009) and COCO (Lin et al., 2014). We observe that *RieM*-based neural compression significantly outperforms existing data-free compression methods and surpasses those requiring additional fine-tuning with real data. For instance, *RieM* allows us to reduce the model size of LeNet-5 by 8.5 times with a 0.18% accuracy improvement on MNIST. For Swin-Transformer, *RieM* achieves a fourfold reduction on the model size with only a 0.2% accuracy degradation on ImageNet-1k, without additional fine-tuning using real data. On a ResNet-based DETR (Carion et al., 2020), *RieM* also results in a 4.8 times reduction in model size with 0.1% improvement on AP_{small} on the COCO object detection benchmark.

Our method represents an effective attempt to interpret neural models using more advanced physical architectures. The core idea is that if we aim to transform neural models into more reasonable physics-inspired dynamical systems, we should focus on the interactions between neurons. By expressing these interactions with more sophisticated physical notions, such as Riemannian metrics, we can achieve more parameter-efficient neural representations. Experimental results further validate the superiority of our method in vision tasks. Codes are publicly available[1].

[1] https://github.com/pzqpzq/flat-learning

2. Preliminaries

2.1. Metric-based Neuronal Dynamics

A neural model is composed of neural layers interpreted as weight matrices. Suppose we have a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with $m, n \in \mathbb{N}^+$, then each neural weight $\mathbf{W}_{xy} \equiv \mathbf{W}[x, y] \in \mathbb{R}$ where $x \in [1, m]$ and $y \in [1, n]$ can be interpreted as the dynamical interaction between an input neuron x and an output neuron y. Each neuron can be interpreted as a trainable high-dimensional embedding (Pei & Wang, 2023), *e.g.*, an input neuron x refers to a D_q-dimensional vector $\mathbf{q}_x^{(in)} \in \mathbb{R}^{D_q}$ with $D_q \in \mathbb{N}^+$, akin to word embeddings, which are randomly initialized trainable parameters. Therefore, the interaction between neurons corresponds to the dynamical relation computed as

$$\mathbf{W}_{xy} = g(\mathbf{q}_x^{(in)}, \mathbf{q}_y^{(out)}), \tag{1}$$

where g refers to a metric function that takes two D_q-dimensional vectors as input and returns a scalar as output. In this setting, we transform traditional weight-based neural models into neuron-based ones, where the trainable parameters are the neurons themselves rather than the neural weights. Apart from interpreting g as a mathematical metric function, we also endow it with certain physical meaning. From the theory of dynamics, g describes the interaction force between two moving objects, *i.e.*, two neurons with changeable neural states in our research context.

2.2. Riemannian Metrics

Before introducing Riemannian metrics, we briefly introduce two essential notions, *i.e.*, smooth manifolds and tangent space, in Riemannian geometry. *Manifolds* are a special class of topological spaces that locally resemble Euclidean space, and the *smooth manifolds* allow for the definition of smooth functions on these spaces, providing a smooth and differentiable structure for geometric and analytical considerations. A *tangent space* to a point on a smooth manifold is a mathematical concept that captures the idea of "instantaneous velocity" or "directionality" at that point.

Now, we regard a *Riemannian metric* as a mathematical structure that endows smooth manifolds with a notion of distance and angles, allowing the definition of geometry on the manifolds. Formally, a *Riemannian metric* g on a smooth manifolds $\mathcal{M}$ is an inner product $g : T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R}$ on each tangent space $T_x\mathcal{M}$ of $\mathcal{M}$ for each $x \in \mathcal{M}$ and

$$g = \sum_{i,j} g_{ij} dx^i \otimes dx^j \tag{2}$$

where $\otimes$ is the tensor product operation that combines two tensors to generate a larger tensor. As the scenario of this paper does not involve sequential data, dx can be simplified to the static variable x for brevity.

In a machine learning context, one can simply regard a Riemannian metric as a function that takes two high-dimensional encodings as input and then returns a scalar that has considered every dimensional component of these input encodings. For example, suppose there is a smooth manifold equipped with a Riemannian metric g, then the relation between two arbitrary d-dimensional points $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ on this manifold can be measured via the pre-defined g. In this case, the metric tensor g is seen as a function that deals with the interactions between every possible pair of dimensional components, *i.e.*,

$$g(\mathbf{u}, \mathbf{v}) = G(\sum_{i,j}^{d} g_{ij}(\mathbf{u}[i], \mathbf{v}[j])) \tag{3}$$

where G and $\{g_{ij}\}$ are specific functions. In the following sections, we will illustrate how to construct computationally efficient functions G and $\{g_{ij}\}$ better suited for neural models, and apply them to model compression.

3. RieM: Neural Riemannian Metrics

Following Sec. 2.1, we interpret a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ as the dynamical interactions between the input neurons $x \in [1, m]$ and the output neurons $y \in [1, n]$. Each neuron refers to a D_q-dimensional dynamical state $\mathbf{q}_x \in \mathbb{R}^{D_q}$ in a neuronal manifolds $\mathcal{Q}$. Our goal is to find a Riemannian metric g such that $g(\mathbf{q}_x, \mathbf{q}_y)$ approximates $\mathbf{W}_{xy}$. Now we obtain the general form of a neural Riemannian metric as $g : T_q\mathcal{Q} \times T_q\mathcal{Q} \to \mathbb{R}$. This form constitutes a group for the neural Riemannian metric, and their collective objective is to transform the high-dimensional neuron interactions into a scalar neural weight. Next, we extend the aforementioned metric form to the D_μ-dimensional metric space $\mathcal{M}$, where each dimension of the metric space corresponds to a neural manifold. As a result, every neural manifold carries an individualized Riemannian sub-metric $g^{(\alpha)}$, $\alpha \in [1, D_\mu]$. Then we can merge them to capture more patterns from each metric dimension, *i.e.*,

$$g(\mathbf{q}_x, \mathbf{q}_y) = \sum_{\alpha=1}^{D_\mu} \rho_\alpha \cdot g^{(\alpha)}(\mathbf{q}_x, \mathbf{q}_y) \in \mathbb{R} \tag{4}$$

where ρ_α is a trainable scalar linked to each metric dimension. Recall that the essence of the neural Riemannian metrics lies in the comprehensive interaction between the dimensions of the two vectors serving as inputs, *i.e.*, every dimension of $\mathbf{q}_x$ and $\mathbf{q}_y$ should exhibit thoroughly mutual interaction. To achieve this effect, we design two options to construct the analytical form of the sub-metric $g^{(\alpha)}$.

Option One concatenates $\mathbf{q}_x$ and $\mathbf{q}_y$ into a unified vector, which is then fed into a trainable universal approximator, *e.g.*, a feed-forward neural network (FFN),

$$g^{(\alpha)}(\mathbf{q}_x, \mathbf{q}_y) = \text{FFN}\big([\mathbf{q}_x; \mathbf{q}_y]\big) \tag{5}$$

where the FFN maps $\mathbb{R}^{2D_q}$ into a scalar $\mathbb{R}$. This operation can implicitly capture the interactions between the $2D_q$ dimensional components of $\mathbf{q}_x$ and $\mathbf{q}_y$, respectively.

Option Two adopts a more explicit approach to capture the interaction between distinct dimensions (see Figure 1). We begin by constructing D_q different displacement vectors $\mathbf{d}^{(s)} \in \mathbb{R}^{D_q}$ with displacement steps $s = [1, D_q]$ such that,

$$\mathbf{d}^{(s)}[i] = \mathbf{q}_x[i] - \mathbf{q}_y[i+s] \tag{6}$$

Note that we assume that $\mathbf{q}[i] = \mathbf{q}[i-D_q]$ if $D_q < i < 2D_q$. Each displacement vector $\mathbf{d}^{(s)}$ is projected into the metric space through a trainable projection $\mathbf{U}^{(s)} \in \mathbb{R}^{D_q \times D_\mu}$, resulting in D_q distinct D_μ-dimensional metric vectors. Then we use a non-linear summation operation to collapse them into a single D_μ-dimensional metric vector, where each dimension corresponds to a sub-metric,

$$[g^{(1)}, ..., g^{(D_q)}](\mathbf{q}_x, \mathbf{q}_y) = \sum_{s=1}^{D_q} \sigma\left(\mathbf{d}^{(s)}\mathbf{U}^{(s)}\right) \tag{7}$$

where σ is a nonlinear activation function, *e.g.*, *sigmoid* operator. This approach can explicitly capture the relationships between the D_q-dimensional components of $\mathbf{q}_x$ and $\mathbf{q}_y$. However, the current form of Option Two is not suitable for large-scale parallel computation. To vectorize Option Two, we introduce the *displacement matrix* $\Phi^{(s)}$ as follows,

$$\Phi^{(s)}[x, y, i] = \mathbf{q}_x[i] - \mathbf{q}_y\left[i + \lceil s \cdot \frac{D_q}{D_s} \rceil\right] \tag{8}$$

where $D_s \in \mathbb{N}^+ < D_q$. Thus, Eq. 7 vectorizes as

$$g^{(\alpha)}(\mathbf{q}_x, \mathbf{q}_y) = \sum_{s=1}^{D_s} \sigma\left(\Phi^{(s)}\mathbf{U}^{(s)}\right) \tag{9}$$

We observe that, under the same parameter scale, the hidden layer width of the metric model obtained by Option Two is significantly larger than that of Option One. Furthermore, by utilizing the displacement matrix, Option Two can incorporate as much complete information as possible for $\mathbf{q}_x$ and $\mathbf{q}_y$. Consequently, Option Two demonstrates a stronger capacity to approximate non-linear relationships than Option One. This conclusion aligns with the mathematical validation (Appendix D) and the empirical results (Fig. 2 and Table 5), where the fitting error of Option Two is significantly lower than that of Option One. Therefore, we choose Option Two as the analytical form of the sub-metric $g^{(\alpha)}$ in Eq. 4. Substituting Eq. 9 into Eq. 4, we have

$$g(\mathbf{q}_x, \mathbf{q}_y) = \sum_{\alpha=1}^{D_\mu} \rho_\alpha \cdot \sum_{s=1}^{D_s} \sigma\left(\Phi^{(s)}\mathbf{U}^{(s)}\right) \tag{10}$$

Unless otherwise noted, we use Eq. 10 as the analytical form of our proposed *Neural Riemannian Metrics*, *aka.*, *RieM*, in the following sections. By default, each weight matrix is equipped with an individual metric.

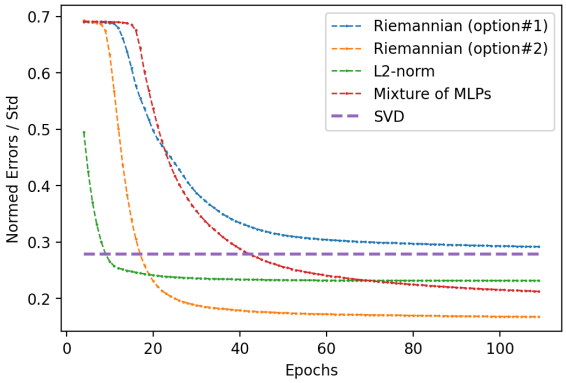

Figure 2: We extract a weight matrix of $\mathbb{R}^{128\times256}$ from the pre-trained ResNet-50. Using $128 + 256$ neurons with 40-dimensional dynamical states each, we reconstruct the matrix based on different metrics, thereby reducing the parameter count of the matrix to approximately 30% of its original size. We also employ the SVD method to compress the matrix at the same compression rate. The loss values between the compressed matrices obtained by each method and the original matrix are recorded during iterations.

4. Neural Compression in Metric Space

In Section 3, we introduced a Riemannian metric for neural manifolds to interpret the neural weights between two neurons as interactions between their dynamical states. In line with (Pei & Wang, 2023), this mechanism can significantly reduce the parameter size of any neural model by transforming the trainable parameters from neural weights to neuron states, which are trainable high-dimensional embeddings. For an arbitrary weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, we expect to obtain a *RieM* metric g proposed by Eq. 10 and a set of neuronal dynamical states, such that $\mathbf{W}_{xy}$ can be approximated as $g(\mathbf{q}_x^{(in)}, \mathbf{q}_y^{(out)})$. We denote the currently reconstructed weight matrix as $\hat{\mathbf{W}}$. We then take the original weight matrix $\mathbf{W}$ as the target and backpropagate the error $\|\hat{\mathbf{W}} - \mathbf{W}\|_2^2$. We iteratively update the neuronal dynamical states and the trainable parameters in the Riemannian metric function. By adjusting the numerical values of the dimensions D_q for each neuron, we can reduce the total number of parameters to reconstruct the weight matrix $\mathbf{W}$ from mn to $D_q \cdot (m+n) + D_\mu \cdot (1 + D_s D_q)$, where $D_\mu D_s D_q \ll m+n$, while preserving its intrinsic patterns as more as possible.

We have discussed how to compress individual neural weight matrices one by one using the Riemannian metric. Next, we will explore how to compress multiple neural weight matrices synchronously. Indeed, we observe that concatenating multiple neural weight matrices according to a specific topological relationship and then compressing them as a whole using the Riemannian metric can effectively reduce compression losses. This is because weight matrices

Table 1: Comparison of neural models compressed by different quantization methods on ImageNet classification benchmark. We abbreviate the numbers of weight/activation bits as W/A-bit. The SIZE indicates the number of bits of full model storage. We also attempt to replace the proposed *RieM* metric with a simple L_p-norm, where the specific value of $p = \{1, 2, 3\}$ is determined by its performance on the development set.

	METHOD	DATA-FREE	SIZE (MB)	W/A-BIT	TOP-1 (%)
RESNET-18	ORIGINAL	×	46.83	32/32	71.47
	DFQ (NAGEL ET AL., 2019)	√	8.36	6/6	66.30
	UDFC (BAI ET AL., 2023)	√	8.36	6/6	**72.70**
	RIEM (OURS)	√	8.36	8/16	71.80
	DDAQ (LI ET AL., 2022C)	√	5.58	4/4	58.44
	DSG (ZHANG ET AL., 2021)	×	5.58	4/4	34.33
	UDFC (BAI ET AL., 2023)	√	5.58	4/4	63.49
	LP-NORM (PEI & WANG, 2023)	√	5.58	8/16	64.52
	RIEM (OURS)	√	5.58	8/16	**66.30**
RESNET-50	ORIGINAL	×	102.53	32/32	77.72
	OSME (CHOUKROUN ET AL., 2019)	√	12.28	4/32	67.36
	GDFQ (XU ET AL., 2020)	×	12.28	4/4	55.65
	SQUANT (GUO ET AL., 2022)	√	12.28	4/4	70.80
	UDFC (BAI ET AL., 2023)	√	12.28	4/4	72.09
	LP-NORM (PEI & WANG, 2023)	√	12.28	8/16	72.96
	RIEM (OURS)	√	12.28	8/16	**73.26**
DENSENET-121	ORIGINAL	×	32.34	32/32	74.36
	OMSE (CHOUKROUN ET AL., 2019)	√	6.00	4/32	64.40
	UDFC (BAI ET AL., 2023)	√	6.00	4/32	70.15
	LP-NORM (PEI & WANG, 2023)	√	6.00	8/16	71.66
	RIEM (OURS)	√	6.00	8/16	**73.15**

in neural models are low-rank matrices, sharing correlated features with each other. As the number of concatenated matrices increases, the shared correlated features also increase, leading to better compression quality. To save time, we use the SVD method to pre-compress these weight matrices with different combinations of concatenation methods.

We define the Shared Correlation Counts (SCC) for a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with $n \geq m$ to estimate the number of inherent correlated features it may contain, *i.e.*,

$$\Omega(\mathbf{W}) = \sum_{r=1}^{n} \frac{m}{\|\mathcal{S}^{(r)}(\mathbf{W}) - \mathbf{W}\|_2^2} \tag{11}$$

where $\mathcal{S}^{(r)}(\mathbf{W}) = \mathbf{U}[:, :r] \cdot \mathbf{\Sigma}[:r, :r] \cdot \mathbf{V}^\top[:r, :]$ refers to the reduced matrix via the SVD method. Based on SCC, we are capable of determining a structurally optimal topological arrangement to reduce the overall compression loss. In general, a higher SCC value indicates that the matrix is more challenging to compress.

Given a pre-trained neural model with L neural weight matrices, denoted as $\mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)}$, we compute their respective SCC values, obtaining $\Omega(\mathbf{W}^{(1)}), ..., \Omega(\mathbf{W}^{(L)})$. We then sort these values in ascending order as a sorted sequence and label the corresponding sorted weight matrices as $\mathbf{W}^{(S_1)}, ..., \mathbf{W}^{(S_L)}$. Subsequently, we concatenate the matrices $\mathbf{W}^{(S_1)}$ and $\mathbf{W}^{(S_L)}$ if they are of similar shapes, padding with zeros in areas where shapes differ. We calculate the SCC value of the resulting concatenated matrix. If this value exceeds $\frac{1}{2}(\Omega(\mathbf{W}^{(S_1)}) + \Omega(\mathbf{W}^{(S_L)}))$, we include this new concatenated matrix in the sorted sequence, replacing the originial $\mathbf{W}^{(S_1)}$ and $\mathbf{W}^{(S_L)}$, which are then removed from the sequence. Iterating through this process for each original neural weight matrix yields L^* matrices, where $L^* \leq L$. As a result, we only need to apply compression methods based on the neural Riemannian metric to these L^* matrices to deal with more shared correlated features, thus superior compression efficiency can be achieved.

5. Accelerating Metric-based Neural Inference

While we can significantly reduce the storage size of neural models, we still need to address how to reduce the computational complexity. According to Sec. 4, we can equivalently represent the neural weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ as a distance matrix composed of $m + n$ points in D_q dimensions, using the Riemannian metric g as the distance function. Utilizing a fast algorithm (Indyk & Silwal, 2022) specifically designed for distance matrices, the multiplication of a metric-based weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with any vector $\mathbf{y} \in \mathbb{R}^m$ can be performed with a reduced spatial complex-

Table 2: Comparison of neural models compressed by different pruning methods on ImageNet classification benchmark. W-bit denotes the bit-width of weights.

	METHOD	PRUNE-RATIO	W-BIT	SIZE (MB)	FLOPS (G)	TOP-1 (%)
RESNET-34	ORIGINAL	0%	32	87.32		73.27
	NEURON MERGE (KIM ET AL., 2020)	10%	32	78.8	6.84	67.10
	UDFC (BAI ET AL., 2023)	10%	6	14.8	6.84	69.86
	RIEM (OURS)	10%	6	14.8	5.30	**72.216**
	NEURON MERGE (KIM ET AL., 2020)	30%	32	61.6	5.30	39.40
	UDFC (BAI ET AL., 2023)	30%	6	11.6	5.30	59.25
	RIEM (OURS)	30%	6	11.6	5.30	**70.144**
RESNET-101	ORIGINAL	0%	32	178.81		77.31
	NEURON MERGE (KIM ET AL., 2020)	10%	32	154.4	3.24	72.46
	UDFC (BAI ET AL., 2023)	10%	6	28.8	3.24	74.69
	RIEM (OURS)	10%	6	28.8	2.52	**76.032**
	NEURON MERGE (KIM ET AL., 2020)	30%	32	112.4	2.52	38.44
	UDFC (BAI ET AL., 2023)	30%	6	21.2	2.52	65.76
	RIEM (OURS)	30%	6	21.2	2.52	**73.296**

ity of $O((m+n)D_q)$. However, the time complexity of this computation still remains $O(mn)$.

To further decrease the time complexity of the $RieM$-based model structure, we propose to eliminate redundant computations between the resulting dynamical states of neurons based on their topological sturctures. The core idea is that if the dynamical states of some neurons are sufficiently close, we can merge them into a single neuron, thereby collapsing their respective operations into a single operation. We name this approach as a dynamical merging mechanism for neurons, *i.e.*, *dyMerge*. Recall that we have m input neurons $\mathbf{q}_1^{(in)}, ..., \mathbf{q}_m^{(in)}$, and n output neurons $\mathbf{q}_1^{(out)}, ..., \mathbf{q}_n^{(out)}$ to construct a weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$. Given a vector $\mathbf{S}^{(in)} \in \mathbb{R}^m$, we define $\mathbf{S}^{(out)} = \mathbf{W}^\top \mathbf{S}^{(in)} \in \mathbb{R}^n$ to simulate the inference process. Using our proposed *RieM* to interpret each weight value in $\mathbf{W}$, we have

$$\mathbf{S}^{(out)}[x] = \sum_{i=1}^{m} g(\mathbf{q}_i^{(in)}, \mathbf{q}_x^{(out)}) \cdot \mathbf{S}_i^{(in)} \tag{12}$$

where g refers to the Riemannian metric in Eq. 10. For each weight matrix, we construct an MLP $\mathcal{N} : \mathbb{R}^{D_q} \times \mathbb{R}^{D_q} \to \mathbb{R}$ that maps the dynamic state of two output neurons to their output correlation coefficient, denoted as $\epsilon_{xy} = \mathcal{N}(\mathbf{q}_x^{(out)}, \mathbf{q}_y^{(out)})$. It is worth noting that the network $\mathcal{N}$ takes only those neurons as input whose *neuronal dissimilarity*, calculated as $\|\mathbf{q}_x^{(out)} - \mathbf{q}_y^{(out)}\|_2$, is smaller than a specific merging threshold. The network $\mathcal{N}$ uses randomly generated $\mathbf{S}^{(in)} \in \mathbb{R}^m$ as the training input and employs $\mathbf{S}^{(out)}$ obtained from Eq. 12 as the labels, *i.e.*,

$$\arg\min_{\mathcal{N}} \left\| \epsilon_{xy} - \frac{\mathbf{S}^{(out)}[x]}{\mathbf{S}^{(out)}[y]} \right\|_2^2 \tag{13}$$

During iterative updating the dynamic states of neurons and metric functions, training for the network $\mathcal{N}$ is randomly implemented. We obtain the output correlation coefficients between output neurons once the training is done. At this point, there is no need to retain the specific parameters of the network $\mathcal{N}$, since it is sufficient to preserve the output correlation coefficients for each pair of neurons with neuronal dissimilarity below the merging threshold.

Now we can determine the model storage size, computational complexity and inference accuracy, by adjusting the neurons' dynamical dimension D_q and the merging threshold. Although lower values of D_q or the merging threshold result in reduced storage size and computational complexity, they also lead to lower inference accuracy. However, these factors are not linearly correlated, and it is unnecessary to simultaneously improve all indicators of the model. Instead, a specific indicator can be chosen as the global objective, while other indicators only need to meet specific constraints. In the experiments, the global objective is to maximize the model's inference accuracy on the validation set. We adjust the neural dimensions D_q and the merging threshold, ensuring that the model's storage size and computational complexity are equivalent to the baselines for comparison.

6. Experiments

6.1. Summary and Datasets

The experimental section primarily evaluates the data-free compression effectiveness of our proposed method on visual models. We first compare our method with existing data-free compression techniques (Bai et al., 2023) based on quantization and pruning respectively on the ImageNet-1k dataset (Deng et al., 2009). Then we compare with compression methods designed for the Visual Transformer (Liu et al., 2021). Experiments are also performed on more complex

Table 3: Comparison of the Vision Transformers compressed by different quantization methods on ImageNet classification benchmark. We abbreviate bit operations as BOPs, and the numbers of weight/activation bits as W/A-bit. Standard and Standard V2 are typical quantization methods, whose implementation details are presented in Li et al. (2022b) and Li et al. (2023), respectively.

	METHOD	DATA-FREE	W/A-BIT	SIZE (MB)	BOPS (G)	TOP-1(%)
SWIN-T	ORIGINAL	×	32/32	116	4608	81.35
	STANDARD	×	4/8	14.5	144	70.16
	STANDARD V2	×	4/8	14.5	144	75.51
	PSAQ-VIT	√	4/8	14.5	144	71.79
	PSAQ-VIT V2	√	4/8	14.5	144	76.28
	RIEM (OURS)	√	4/8	14.5	144	76.30
	PSAQ-VIT V2	√	8/8	29	288	80.21
	RIEM (OURS)	√	8/8	29	288	**80.85**
SWIN-S	ORIGINAL	×	32/32	200	8909	83.20
	STANDARD	×	4/8	25	278	73.33
	STANDARD V2	×	4/8	25	278	78.22
	PSAQ-VIT	√	4/8	25	278	75.14
	PSAQ-VIT V2	√	4/8	25	278	78.86
	RIEM (OURS)	√	4/8	25	278	79.84
	PSAQ-VIT V2	√	8/8	50	557	82.13
	RIEM (OURS)	√	8/8	50	557	**82.96**

Table 4: Comparisons with DETR-based methods on the COCO object detection benchmark. We use ResNet-50 as the backbone model. We report bbox AP (Average Precision) on the validation dataset and present the variation in performance metrics of various methods compared to the original model. Quant-DETR and SVD-DETR refer to the compression methods based on typical quantization and Singular Value Decomposition.

METHOD	DATA-FREE	W-BIT	SIZE (MB)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
DETR	×	32	159.0	40.1	60.6	42.0	18.3	43.3	59.5
T-DETR (ZHEN ET AL., 2022)	×	8	43.6	-0.6	-0.8	-0.4	**+0.5**	-0.9	**-1.5**
T-DETR	×	4	33.4	-2.2	-2.7	-2.2	-1.0	-2.7	-3.2
QUANT-DETR	√	8	43.6	-2.2	-1.2	-3.1	-2.5	-2.5	-1.8
SVD-DETR	√	8	33.4	-11.5	-14.2	-12.8	-6.1	-15.1	-11.6
RIEM-DETR (OURS)	√	8	43.6	**-0.4**	-0.6	**+0.1**	+0.4	**-0.3**	**-1.5**
RIEM-DETR (OURS)	√	8	33.4	-0.7	**-0.5**	-1.2	+0.1	-1.3	-2.1
RIEM-DETR (OURS)	√	8	26.7	-2.8	-2.5	-3.4	-2.4	-4.4	-4.1

model structures focused on the COCO object detection dataset (Lin et al., 2014). We also conduct preliminary experiments in non-data-free scenarios, involving training of randomly initialized neural structures rather than pre-trained ones, on datasets such as MNIST (Deng, 2012) and CIFAR-100 (Krizhevsky et al., 2009). Furthermore, information retrieval experiments are conducted on the ImageNet dataset, with comparisons against mainstream dimensionality reduction methods.

6.2. Implementation Details

The implementation of our *RieM*-based compression is straightforward: expressing each neural weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ of a pre-trained model as $m+n$ neural dynamic states of dimension D_q through *RieM* (Eq. 10), *i.e.*, $\mathbf{q}_1^{(in)}, ..., \mathbf{q}_m^{(in)}; \mathbf{q}_1^{(out)}, ..., \mathbf{q}_n^{(out)}$. This allows us to approximate each entry $\mathbf{W}_{xy} \in \mathbb{R}$ as $g(\mathbf{q}_x^{(in)}, \mathbf{q}_y^{(out)})$ to obtain a *RieM*-based model. Then we compress this *RieM*-based model using the method elaborated in Sec. 4, followed by the method proposed in Sec. 5 to reduce the computational complexity during the inference process. We quantize both the neuronal dynamical states, *e.g.*, $\mathbf{q}_x \in \mathbb{R}^{D_q}$, and the computed metric, *i.e.*, $g(\mathbf{q}_x^{(in)}, \mathbf{q}_y^{(out)})$.

For each pre-trained model to be compressed, we randomly select ten sets of seeds, yielding ten different randomly initialized *RieM*-based models, including trainable neural dynamics states and the Riemannian metrics for each weight matrix. We first adjust hyperparameters, such as the neuronal dimension D_q, the internal dimension D_μ of the Riemannian metrics, ensuring that the resulting *RieM*-based

models have storage model size and computational complexity (measured in FLOPs or BOPs) consistent with various baselines. We then implement data-free training on the neural states and metric functions using the original weight matrices of the pre-trained model as the target, while knowing nothing about the real datasets. After this data-free training, we select the top three performing models based on their performance on the development set and average their evaluations on the test set as the final results.

All experiments can be conducted on a single 24GB memory GeForce RTX 4090 GPU, and the total time required to convert a pre-trained model, *e.g.*, Swin-T, to a trained *RieM*-based model typically ranges from a few hours to several days, depending on the desired model performance. The higher the performance requirements for the model, the more configurations need to be explored, resulting in longer actual processing times.

6.3. Main Results

For the experiments on ImageNet-1k, we mainly compare our methods with UDFC (Bai et al., 2023), which is a unified data-free compression framework that performs data-free pruning and quantization simultaneously without additional fine-tuning on real datasets. We first compare our method with existing quantization methods (see Table 1). The empirical results show that converting the pre-trained visual models to the $RieM$-based models can significantly reduce the storage model size while maintaining the inference accuracy. For example, with an identical storage size on ResNet-50, our method improves the model's inference accuracy by 2.2% compared to the data-free quantization method SQuant (Guo et al., 2022) and by 1.2% compared to the UDFC. We also observe that our method even outperform the non data-free quantization methods that require fine-tuning on real-data.

Then we compare our method with existing pruning methods, with which our main competitors involve Neuron Merging (NM) (Kim et al., 2020), which is a one-to-one compensation method, and UDFC. Table 2 shows that our data-free metrics-based compression methods outperforms the pruning baselines. The experimental results demonstrate that our data-free compression method significantly outperforms the baselines. It achieves higher inference accuracy with a lower model storage size. Taking ResNet-101 as an example, under a 30% pruning ratio, the *RieM*-based compression method improves the model's inference accuracy by 34.8% compared to the Neuron Merge method and by 7.5% compared to the UDFC method.

To further validate the performance of our approach, we also conduct experiments specifically targeting the Vision Transformer architecture. The primary competitor in this part is a data-free compression method tailored for Vision Transformer, namely PSQA-ViT (Li et al., 2022b), and its successor PSQA-ViT V2 (Li et al., 2023). These method employ a relative value metric called patch similarity to generate data from pre-trained vision transformers. As presented in Table 3, we observe that despite our approach lacking a tailored design for the Vision Transformer, it still outperforms PSQA-ViT V2 in terms of model inference accuracy under equivalent storage size and computational complexity conditions.

We also conduct experiments on the COCO object detection dataset. We choose the DETR model based on ResNet-50 as the target for compression. Our primary competitor is T-DETR (Zhen et al., 2022), a tensorized detection transformer achieved by reshaping weight matrices into high-dimensional tensors and employing low-rank tensor factorization. Evaluation metrics include Average Precision (AP) with different Intersection over Union (IoU) thresholds, and AP for objects with varying scales, all presented in Table 4. The experimental results indicate that, under equal storage sizes, among the six AP metrics, except for AP_S, data-free $RieM$-based models outperform the T-DETR model, which requires fine-tuning with real data.

Comparison with typical dimensionality reduction methods is presented in Appendix B. Ablation study is presented in Table 5 of Appendix A.

6.4. Flexibility in Compressed Model Size

By adjusting the embedding dimension of neurons and constructing *RieM*, our method enables the compression of neural models to any target size. Moreover, merging adjacent neurons can further reduce the model's size.

In order to compress the model to a specific ratio of its original size, we need to compute the required embedding dimensions for neurons in each neural layer. Suppose a neural layer has m input neurons and n output neurons; then, the parameter count for that layer is $m \cdot n$. If we intend to compress it to one-tenth of its original size, the dimension d for each neuron's embedding is approximately $\frac{mn}{10 \cdot (m+n)}$. Typically, $mn \gg m + n$; otherwise, we can merge adjacent neurons or adjust the compression ratios of other neural layers to ensure the expected model compression.

As in Figure 3, we conduct preliminary experiments on Swin-T using ImageNet-1k. Note that there is still room for improvement in accuracy here. By employing different compression strategies and determining which neurons in which layers should have higher or lower embedding dimensions for the target model size, we can obtain different compressed models with varying performance.

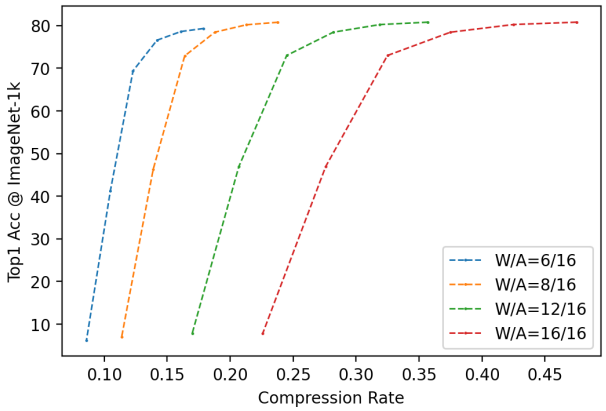

Figure 3: We convert a pretrained ViT-T model into a *RieM*-based form by adjusting the dimensions of individual neurons such that the model size of the *RieM*-based model equaled a target value. Notably, for the same target model size, the dimensions of neurons in different layers can have varying values, meaning that different configurations, even with the same model size, may result in different accuracies. In this experiment, we randomly select three different configurations for each target model size and compute the average of their resulting accuracies as the outcome. We observe that using lower bit quantization leads to a higher cost-effectiveness of accuracy versus model size, which also imposes a lower accuracy ceiling.

6.5. Bilinear and Symmetric Conditions

The original *RieM* presented in Eq. 6 and 7 is computed based on the relationships between all dimensions of two vectors. Even if we exchange the vectors, the dimension-pairs they generate and the results computed based on *RieM* remain unchanged, thus satisfying symmetry. Therefore, the original *RieM* satisfies the symmetric property defined in a typical Riemannian metric. However, if we simplify the original version of *RieM* (Eq. 8 and 9) by not computing all dimension-pairs between the two vectors but only a subset in a "skipping" manner, then this simplified *RieM* may not fully satisfy symmetry.

As presented in Table 5, we observe that both the original (*no-skipping*) and simplified *RieM*s yield consistent performance in model compression tasks. Furthermore, we replace the activation function σ within *RieM* with partially linear functions (*e.g.*, ReLU) and initialize the embedding appropriately to ensure *RieM* satisfies bilinearity. Although the resulting *RieM*, which satisfies bilinearity and symmetry, performs equally well in data-free compression tasks as the simplified *RieM*, it requires an extra implementation to guarantee its bilinearity and symmetry. This fact suggests that the metric used to describe the relationships between neurons does not necessarily need to satisfy symmetry and bilinearity when placing neural models in neuronal space. Introducing specific specialized metrics (such as a skew-symmetric or anti-symmetric form used in theoretical physics) can lead to more efficient neural state spaces.

7. Related Work

In the realm of accelerating model inference, researchers have explored network pruning (Liu et al., 2018; Blalock et al., 2020) and quantization methods (Yang et al., 2019; Nagel et al., 2021), which typically require fine-tuning with data. However, privacy and security concerns often make fine-tuning impractical for applications with sensitive data. To address this, it is necessary to consider data-free compression, which mainly involves three categories as follows.

Data-free pruning eliminates reliance on the fine-tuning process with real-world dataset. Two methods are employed: data-free parameter pruning (Srinivas & Babu, 2015) identifies redundant neurons, and Neuron Merging (Kim et al., 2020) extends the method to convolutional layers.

Data-free quantization introduced by (Nagel et al., 2019) faces performance drops at low bit-widths, prompting recent studies like ZeroQ (Cai et al., 2020), IntraQ (Zhong et al., 2022), and DSG (Zhang et al., 2021) to use generator architectures, generating synthetic samples to replace the original data.

Unified data-free compression such as Bai et al. (2023) assumes that partial information from a damaged channel is preserved via a linear combination of other channels, concurrently prunes and quantizes without real data.

8. Conclusion

In this paper, we investigate the integration of Riemannian dynamics with neural structures to develop a more effective data-free neural compression method. We introduce a Riemannian metric for neural models, denoted as *RieM*, to interpret neural weights as the relations between the dynamic states of neurons, resulting in a reduced parameter size and lower computational complexity for the *RieM*-based neural model. Experimental results on visual datasets, such as MNIST, CIFAR-100, ImageNet-1k, COCO object detection, demonstrate that our approach outperforms existing data-free compression methods in neural compression, surpassing even those requiring external real data for fine-tuning. Future work involves refining the computational form of Riemannian metrics, reducing the conversion time, and deriving a more accurate physics-inspired framework to further enhance the representation and computational efficiency of neural structures.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

This work was supported in part by the National Key R&D Program of China under Grant 2023YFC2508704, in part by National Natural Science Foundation of China: 62236008, U21B2038 and 61931008, and in part by the Fundamental Research Funds for the Central Universities.

References

Bai, S., Chen, J., Shen, X., Qian, Y., and Liu, Y. Unified data-free compression: Pruning and quantization without fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5876–5885, 2023.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13169–13178, 2020.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European conference on computer vision*, pp. 213–229. Springer, 2020.

Choukroun, Y., Kravchik, E., Yang, F., and Kisilev, P. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3009–3018. IEEE, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Guo, C., Qiu, Y., Leng, J., Gao, X., Zhang, C., Liu, Y., Yang, F., Zhu, Y., and Guo, M. Squant: On-the-fly data-free quantization via diagonal hessian approximation. *arXiv preprint arXiv:2202.07471*, 2022.

Indyk, P. and Silwal, S. Faster linear algebra for distance matrices. *Advances in Neural Information Processing Systems*, 35:35576–35589, 2022.

Kaul, P. and Lall, B. Riemannian curvature of deep neural networks. *IEEE transactions on neural networks and learning systems*, 31(4):1410–1416, 2019.

Kim, W., Kim, S., Park, M., and Jeon, G. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33:585–595, 2020.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lee, J. M. and Lee, J. M. *Smooth manifolds*. Springer, 2012.

Li, Q., Lin, T., and Shen, Z. Deep learning via dynamical systems: An approximation perspective. *Journal of the European Mathematical Society*, 25(5):1671–1709, 2022a.

Li, Z., Ma, L., Chen, M., Xiao, J., and Gu, Q. Patch similarity aware data-free quantization for vision transformers. In *European Conference on Computer Vision*, pp. 154–170. Springer, 2022b.

Li, Z., Ma, L., Long, X., Xiao, J., and Gu, Q. Dual-discriminator adversarial framework for data-free quantization. *Neurocomputing*, 511:67–77, 2022c.

Li, Z., Chen, M., Xiao, J., and Gu, Q. Psaq-vit v2: Toward accurate and general data-free quantization for vision transformers. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Liu, G.-H. and Theodorou, E. A. Deep learning theory review: An optimal control and dynamical systems perspective. *arXiv preprint arXiv:1908.10920*, 2019.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Nagel, M., Baalen, M. v., Blankevoort, T., and Welling, M. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1325–1334, 2019.

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., Van Baalen, M., and Blankevoort, T. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Pei, Z. and Wang, S. Dynamics-inspired neuromorphic visual representation learning. In *International Conference on Machine Learning*, pp. 27521–27541. PMLR, 2023.

Petersen, P. *Riemannian geometry*, volume 171. Springer, 2006.

Ramer, U. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

Shao, H., Kumar, A., and Thomas Fletcher, P. The riemannian geometry of deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 315–323, 2018.

Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

Udriste, C. and Tevy, I. Geometric dynamics on riemannian manifolds. *Mathematics*, 8(1):79, 2020.

Xu, S., Li, H., Zhuang, B., Liu, J., Cao, J., Liang, C., and Tan, M. Generative low-bitwidth data free quantization. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pp. 1–17. Springer, 2020.

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., and Hua, X.-s. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

Zhang, X., Qin, H., Ding, Y., Gong, R., Yan, Q., Tao, R., Li, Y., Yu, F., and Liu, X. Diversifying sample generation for accurate data-free quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15658–15667, 2021.

Zhen, P., Gao, Z., Hou, T., Cheng, Y., and Chen, H.-B. Deeply tensor compressed transformers for end-to-end object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4716–4724, 2022.

Zhong, Y., Lin, M., Nan, G., Liu, J., Zhang, B., Tian, Y., and Ji, R. Intraq: Learning synthetic images with intra-class heterogeneity for zero-shot network quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12339–12348, 2022.

A. Ablation Study

Table 5: *RieM* (Option1) refers to Eq. 5 that concatenates the neuronal states into a feedforward neural network. *RieM* (Option2) refers to Eq. 10. SCC refers to the approach (Sec. 4) that merge relevant matrices to reduce compression loss. $DyMerge$ refers to the accelerating mechanism (Sec. 5) that reduces duplicated computations based on neuronal topology.

	METRIC	SCC	DYMERGE	SIZE (MB)	BOPS (G)	TOP-1 (%)
SWIN-T	RIEM (OPTION2)	√	√	29	288	80.85
	RIEM (OPTION2)	×	√	29	288	79.50
	RIEM (OPTION2)	√	×	29	288	80.25
	RIEM (OPTION1)	√	√	29	288	75.38
	SVD	√	×	58	288	76.23
	L1-NORM	√	√	29	288	79.04
	L2-NORM	√	√	29	288	78.24
	L3-NORM	√	√	29	288	78.31
	NO-SKIPPING	√	√	29	288	80.50
SWIN-S	RIEM (OPTION2)	√	√	50	557	82.96
	RIEM (OPTION2)	×	√	50	557	82.41
	RIEM (OPTION2)	√	×	50	557	81.98
	RIEM (OPTION1)	√	√	50	557	79.84
	SVD	√	×	100	557	80.37
	L1-NORM	√	√	50	557	80.96
	L2-NORM	√	√	50	557	81.54
	L3-NORM	√	√	50	557	81.50
	NO-SKIPPING	√	√	50	557	82.85

B. Comparison with Dimensionality Reduction Methods

Table 6: Normalized matrix-vector production error on the squared ImageNet dataset. The ratio $T_{comp.}/T_{naive}$ represents the degree of speeding up for the current method with the naive one, *i.e.*, direct matrix-vector production, by setting different hyper-parameters.

MATRIX SHAPE	$\mathbb{R}^{1000\times1000}$		$\mathbb{R}^{5000\times5000}$		$\mathbb{R}^{10000\times10000}$	
$T_{comp.}/T_{naive}$	0.1	0.3	0.1	0.3	0.1	0.3
ISOMETRIC MAPPING	1.22E-01	1.23E-01	6.27E-02	6.28E-02	4.55E-02	4.56E-02
AUTOENCODER	4.60E-02	3.66E-02	1.57E-02	2.86E-02	4.43E-02	4.33E-02
DEEP AUTOENCODER	3.16E-02	3.41E-02	1.35E-02	1.58E-02	9.50E-03	3.93E-02
LOCALLY LE	3.16E-02	3.16E-02	1.41E-02	1.41E-02	9.98E-03	9.98E-03
NYSTROM	3.16E-02	3.16E-02	1.41E-02	1.41E-02	9.98E-03	9.98E-03
KERNEL PCA	1.35E-03	1.35E-03	7.70E-04	7.80E-04	7.40E-04	7.50E-04
LP-NORM	2.50E-04	1.31E-04	2.15E-05	1.65E-05	9.87E-06	7.88E-06
RIEM (OURS)	2.20E-04	1.20E-04	1.58E-05	1.26E-05	5.56E-06	6.27E-06

C. Evaluations on Real-Data-Driven Neural Compression

Table 7: A preliminary evaluation of our proposed metric-based methods on simple models and small-scale datasets. We use randomly initialized models such as LeNet-5 and ResNet-9 and interpret them using our proposed metrics. These models are separately trained on datasets like MNIST and Cifar100, and their Top-1 test accuracies are obtained. For simplicity, we replace only the fully connected (FC) layer with our metric and present their parameter counts.

	METRIC	NO.PARAMS (FC LAYER)	TOP-1 (%)	
			MNIST	CIFAR100
LENET-5	N/A	59.3K	99.10	44.30
LENET-5	L1-NORM	6.3K	98.95	44.35
LENET-5	L2-NORM	6.3K	99.17	44.45
LENET-5	L3-NORM	6.3K	99.10	44.40
LENET-5	RIEM	7.2K	**99.28**	**44.78**
RESNET-9	N/A	102.8K	99.62	67.58
RESNET-9	L1-NORM	32.0K	99.58	67.54
RESNET-9	L2-NORM	32.0K	99.64	67.63
RESNET-9	L3-NORM	32.0K	99.62	67.53
RESNET-9	RIEM	33.4K	99.69	68.12
RESNET-9	RIEM	51.5K	**99.72**	**68.15**

D. Why Option 2 is Better Than Option 1

Given $q_x,\ q_y \in \mathbb{R}^n$ as the neuronal embeddings, and each embedding has a discrete representation such that $q_x[i] \in \{-N \cdot \delta, -(N-1) \cdot \delta, ..., 0, ..., (N-1) \cdot \delta, N \cdot \delta\}$ for any i. Let's define $\Omega(\mathbf{v})$ as the logarithm of the number of permutations of $\mathbf{v}$, referring to the representational capacity of $\mathbf{v}$. Then, for Option 1, we have,

$$\Omega([q_x; q_y]) = (2N)^{2n} \tag{14}$$

For Option 2, we have,

$$\Omega\big(g(q_x, q_y)\big) = (4N)^n \cdot (\gamma \cdot 4N)^n \cdot ... \cdot (\gamma^D \cdot 4N)^n = \prod_{s=0}^{D} (\gamma^s \cdot 4N)^n \tag{15}$$

where $\gamma \in (0, 1]$ refers to the approximated decay between each displacement step. Next, let's define the difference between Option 1 and Option 2 as,

$$\begin{aligned} \varepsilon &= \log\Big(\Omega\big(g(q_x, q_y)\big)\Big) - \log\Big(\Omega([q_x; q_y])\Big) \\ &= (D+1) \cdot n \cdot \log(4N) + \frac{(D+1) \cdot D}{2} \cdot \log(\gamma) - 2n \cdot \log(2N) \\ &= f(N, n, D; \epsilon) \end{aligned} \tag{16}$$

where $\gamma = 1 \cdot 10^{-\epsilon}$. Using approximation or computational validation (see Figure 4), we can conclude that

$$\varepsilon = -(a \cdot D) \cdot \epsilon + b \cdot N + c \cdot n \tag{17}$$

where $a, b, c \approx 1$. Thus, $\varepsilon \leq 0$ holds if and only if

$$\epsilon \geq \frac{N+n}{D} \gg 1 \tag{18}$$

Because each element of neuronal embedding is initialized independently at random, and the interdimensional differences between them are also independent, thus, γ is often slightly less than 1. For sufficiently random initialized neuronal embeddings, γ cannot approach zero infinitely. Therefore, we need not worry about γ being so small that ε becomes negative; that is, the probability that the representational capacity of Option 2 exceeds that of Option 1 approaches 1 infinitely.

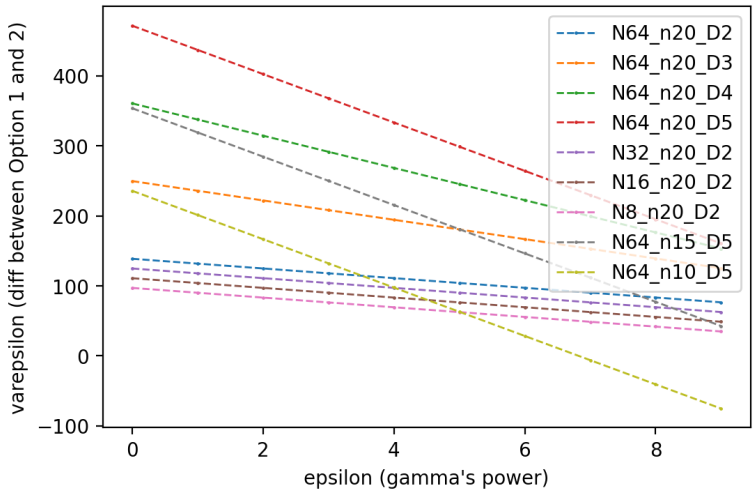

Figure 4

E. Metrics-based Matrix-Vector Product

Utilizing a fast algorithm (Indyk & Silwal, 2022) specifically designed for distance matrices, the multiplication of a metric-based weight matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with any vector $\mathbf{y} \in \mathbb{R}^m$ can be performed with a reduced spatial complexity of $O((m+n)\cdot D_q)$. We first validate the case with a metric function of L_1-norm. Following Indyk & Silwal (2022), we aim to compute $z^{(h)} = \mathbf{D}^{(h)}y$, where $\mathbf{D}^{(h)} \in \mathbb{R}^{m\times n}$ is a distance matrix constructed via L_1-norm on two sets of points $\mathbf{X}^{(h)} \in \mathbb{R}^{m\times d}$ and $\mathbf{Y}^{(h)} \in \mathbb{R}^{n\times d}$. We generalize the fast matrix-vector product for the squared distance matrices to non-squared distance matrices. For each coordinate $k \in [1, m]$, we have

$$z_k^{(h)} = \mathbf{D}_k^{(h)} y = \sum_{i=1}^{n} y_i \|\mathbf{X}_k^{(h)} - \mathbf{Y}_i^{(h)}\|_1 = \sum_{j=1}^{d}\sum_{i=1}^{n} y_i \cdot |\mathbf{X}_k^{(h)}[j] - \mathbf{Y}_i^{(h)}[j]| \tag{19}$$

Let's denote $i \in \pi_j^+$ implies that $\mathbf{X}_k^{(h)}[j] \geq \mathbf{Y}_i^{(h)}[j]$, and $i \in \pi_j^-$ implies that $\mathbf{X}_k^{(h)}[j] < \mathbf{Y}_i^{(h)}[j]$, then the inner sum of the right side can be expanded as follows:

$$\sum_{i=1}^{n} y_i \cdot |\mathbf{X}_k^{(h)}[j] - \mathbf{Y}_i^{(h)}[j]| = \sum_{i\in\pi_j^+} y_i \cdot (\mathbf{X}_k^{(h)}[j] - \mathbf{Y}_i^{(h)}[j]) + \sum_{i\in\pi_j^-} y_i \cdot (\mathbf{Y}_i^{(h)}[j] - \mathbf{X}_k^{(h)}[j]) \tag{20}$$

The inner sum is then rearranged as follows:

$$\begin{aligned} &\mathbf{X}_k^{(h)}[j]\Big(\sum_{i\in\pi_j^+} y_i - \sum_{i\in\pi_j^-} y_i\Big) + \sum_{i\in\pi_j^-} y_i \mathbf{Y}_i^{(h)}[j] - \sum_{i\in\pi_j^+} y_i \mathbf{Y}_i^{(h)}[j] \\ &= \mathbf{X}_k^{(h)}[j] \cdot (S_Y^+ - S_Y^-) + S_{yY}^- - S_{yY}^+ \end{aligned} \tag{21}$$

where S_Y^+, S_Y^-, S_{yY}^- and S_{yY}^+ are preprocessed terms computed using the partial sum of the sorted array of $\mathbf{X}^{(h)}$ and $\mathbf{Y}^{(h)}$ for each dimension. With the simplest L_1-norm metric function, the corresponding computational complexity will be $O(nd)$ or $O(\max(m,n)d)$, which roughly equals the parameters required to construct the distance matrix.

F. Dimensionality Reduction and Nonlinear Transformation

Our method can provide new insight into dimensionality reduction. As presented in Section 4, a set of vectors $\mathbf{A} = [v_1, ..., v_m]^\top$ can be reconstructed using fewer parameters, *i.e.*, H distinct sets of points $\{\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}; ...; \mathbf{X}^{(H)}, \mathbf{Y}^{(H)}\}$. Unlike traditional dimensionality reduction approaches that explicitly compress an n-dimensional vector to an n^*-dimensional one with $n^* < n$, our method implicitly reduces dimension. For simplicity, we denote $\tilde{\mathbf{X}} \in \mathbb{R}^{m\times d\times H}$

and $\tilde{\mathbf{Y}} \in \mathbb{R}^{n\times d\times H}$ and $\tilde{\mathbf{X}}[i,k,h] = \mathbf{X}_i^{(h)}[k]$. According to the definition of distance matrices, an n-dimensional vector $v_i \subset \mathbf{A}$ can be seen as an array of metrics to the bases $\{\tilde{\mathbf{Y}}_1, ..., \tilde{\mathbf{Y}}_n\}$ as follows:

$$v_i = [\tilde{\mu}(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_1), ..., \tilde{\mu}(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_n)]^\top \tag{22}$$

where the metric function $\tilde{\mu} : \mathbb{R}^{d\times H} \times \mathbb{R}^{d\times H} \mapsto \mathbb{R}$ is a generalized nonlinear metric equipped with finite scalar coefficients $\{\lambda_1, ...\lambda_H\}$ expressed as an analytical form as follows:

$$\tilde{\mu}(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_j) = \sum_{h=1}^{H} \lambda_h \cdot \mu(\tilde{\mathbf{R}}_{ij}^\top[h]) \tag{23}$$

where $\tilde{\mathbf{R}}_{ij} = \tilde{\mathbf{X}}_i - \tilde{\mathbf{Y}}_j$ and μ is L_p-norm; we set $p = 1$ by defaults in practice. As a result, each $v_i \in \mathbb{R}^n$ refers to a point with coordinates $\tilde{\mathbf{X}}_i \in \mathbb{R}^{d\times H}$, and v_i can be approximated as the relations between $\tilde{\mathbf{X}}_i$ and the bases $\{\tilde{\mathbf{Y}}_1, ..., \tilde{\mathbf{Y}}_n\}$. Therefore, if we have a large well-formed $\mathbf{A}$, *i.e.*, $m \gg n > dH$, then when we add a new vector $v_{m+1} \in \mathbb{R}^n$ to $\mathbf{A}$, what we need is to pick an appropriate point $\tilde{\mathbf{X}}_{m+1}$ such that v_{m+1} can be approximated as the relations between $\tilde{\mathbf{X}}_{m+1}$ and the bases via the nonlinear metric $\tilde{\mu}$. In this way, the dimensionality reduction mechanism of our method involves converting a vector into a low-dimensional vector and a set of global bases shared by other vectors while also employing a specific nonlinear metric function. This mechanism extracts the necessary local features and captures more comprehensive global features, ensuring the reduced representation has a hierarchical structure rather than a determined lossy encoding.

Next, let us proceed to the implementation of matrix-vector production. Based on the discussion above, the process of $\hat{z} = \hat{\mathbf{A}}y$ can be interpreted physically. First, the n bases receive signals denoted by $y \in \mathbb{R}^n$, *i.e.*, each basis $\tilde{\mathbf{Y}}_j$ receives a signal $y_j \in y$. Second, a basis $\tilde{\mathbf{Y}}_j$ emits the received signal to the m points, *e.g.*, $\{\tilde{\mathbf{X}}_i \in \mathbb{R}^{d\times H}, i \in [1, m]\}$. Third, the arrival signal at a point $\tilde{\mathbf{X}}_i$ is amplified by a factor $\tilde{\mu}(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_j)$. Finally, the total signals $\hat{z}_i$ received by a point $\tilde{\mathbf{X}}_i$ are as follows:

$$\hat{z}_i = \sum_{j=1}^{n} y_j \cdot \tilde{\mu}(\tilde{\mathbf{X}}_i, \tilde{\mathbf{Y}}_j) \tag{24}$$

In general, Eq. 24 interprets matrix-vector production $\hat{z} = \hat{\mathbf{A}}y$ as follows. Given the nonlinear metric space $\mathcal{SP}(\mathbf{A})$ equipped with a nonlinear metric function $\tilde{\mu}$ over the n bases $\{\tilde{\mathbf{Y}}_1, ..., \tilde{\mathbf{Y}}_n\}$, then $\hat{z} = \hat{\mathbf{A}}y$ is equivalent to the process of signals y transmitting in $\mathcal{SP}(\mathbf{A})$, received by the n points $\{\tilde{\mathbf{Y}}_1, ..., \tilde{\mathbf{Y}}_n\}$ and emitting to the m points $\{\tilde{\mathbf{X}}_1, ..., \tilde{\mathbf{X}}_m\}$.

Theorem F.1. *We can approximate arbitrary matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with the weighted sum of distance matrices $\hat{\mathbf{A}} = \sum_{h=1}^{H} \mathbf{D}^{(h)}$, and H, the number of L_1 distance matrices, is upper-bounded.*

Proof. According to the properties of piecewise linearity and Ramer–Douglas–Peucker algorithm (Ramer, 1972), we can decimate a curve composed of line segments to a similar curve with fewer points. The entries of a matrix can be seen as the lengths of many isolated curves, which can be approximated as the weighted sum of segments, which are the entries of a distance matrix. Moreover, we need at most $\lceil \frac{mn}{d\cdot(m+n)} \rceil$ different distance matrices to cover all the parameters contained in $\mathbf{A}$. For example, an arbitrary matrix with $m \cdot n$ components can be fully expressed as the metrics between a set of $m \cdot n$ 1D points and another set of n or m 1D points. This implies that the upper-bound of H is ceiled $mn/(d \cdot (m + n))$. Therefore, the total number of parameters $H \cdot d \cdot (m + n)$ should not exceed $m \cdot n$. □

Theorem F.2. *Given arbitrary matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $y \in \mathbb{R}^{n\times 1}$, there exists an algorithm to compute $z = \mathbf{A}y$ in $O\big((1 - k \cdot \epsilon) \cdot n^2\big)$, where $k \gg 1$ is a constant related to $\mathbf{A}$, and $\epsilon = \frac{\|z-\hat{z}\|_2}{\sigma_z}$, where σ_z is the standard deviation of z.*

Proof. Recall the element-wise error between $\mathbf{A}$ and the weighted sum of H proper distance matrices is denoted by $\varepsilon_{ij}^{(H)} = \mathbf{A}_{ij} - \sum_{h=1}^{H} \lambda_h \mathbf{D}_{ij}^{(h)}$, then the collection of $\{\varepsilon_{ij}^{(H)},\ i \in [1, m],\ j \in [1, n]\}$ has a mean 0 and standard deviation $\sigma_{\mathbf{A}}^{(H)}$. According to the chain rule, we can conclude that

$$\sigma_{\mathbf{A}}^{(H+1)} = \frac{\sigma_{\mathbf{A}}^{(H)}}{c} \tag{25}$$

where $c > 1$ is a constant related to $\mathbf{A}$. The error related to H is denoted by $\varepsilon_z^{(H)} = \mathbf{A}y - \sum_{h=1}^{H} \lambda_h \mathbf{D}_{ij}^{(h)} y$, then the standard deviation $\sigma_z^{(H)}$ of $\varepsilon_z^{(H)}$ is computed as

$$\sigma_z^{(H)} = \sqrt{\sum_{i=1}^{n} y_i^2 \cdot {\sigma_{\mathbf{A}}^{(H)}}^2} = \sigma_{\mathbf{A}}^{(H)} \cdot \|y\|_2 \tag{26}$$

According to the properties of folded normal distribution, the expectation of $\|z - \hat{z}^{(H)}\|_2$ is proportional to $\sigma_z^{(H)}$. Substituting Eq. 25 and Eq. 26 into the definition of ϵ, we have

$$\epsilon^{(H+1)} = \frac{\|z - \hat{z}^{(H+1)}\|_2}{\sigma_z} = \frac{\zeta \cdot \sigma_z^{(H+1)}}{\sigma_z} = \frac{\zeta \cdot \sigma_{\mathbf{A}}^{(H+1)} \cdot \|y\|_2}{\sigma_z} \tag{27}$$

where ζ denotes a constant determined by the distribution. The ratio of ϵ with different H is given by

$$\frac{\epsilon^{(H+1)}}{\epsilon^{(H)}} = \frac{\sigma_{\mathbf{A}}^{(H+1)}}{\sigma_{\mathbf{A}}^{(H)}} = \frac{1}{c} \tag{28}$$

Obviously, H is proportional to the computational complexity when implementing matrix-vector production. Therefore, Eq. 28 implies that the ratio of computational complexity and ϵ demonstrates linearity with respect to H, which holds for Theorem F.2. Note that the complexity of $O\big((1 - k \cdot \epsilon) \cdot n^2\big)$ indicates that our method is scalable to m. This property is particularly useful when performing neural inference on neural layers with arbitrarily large output widths or dimensions.

□