Orthogonal Over-Parameterized Training

Weiyang Liu[1,2,*] Rongmei Lin[3,*] Zhen Liu[4] James M. Rehg[5] Liam Paull[4] Li Xiong[3] Le Song[5] Adrian Weller[1,6]

[1]University of Cambridge [2]Max Planck Institute for Intelligent Systems [3]Emory University
[4]Mila, Université de Montréal [5]Georgia Institute of Technology [6]Alan Turing Institute [*]Equal Contribution

Abstract

The inductive bias of a neural network is largely determined by the architecture and the training algorithm. To achieve good generalization, how to effectively train a neural network is of great importance. We propose a novel orthogonal over-parameterized training (OPT) framework that can provably minimize the hyperspherical energy which characterizes the diversity of neurons on a hypersphere. By maintaining the minimum hyperspherical energy during training, OPT can greatly improve the empirical generalization. Specifically, OPT fixes the randomly initialized weights of the neurons and learns an orthogonal transformation that applies to these neurons. We consider multiple ways to learn such an orthogonal transformation, including unrolling orthogonalization algorithms, applying orthogonal parameterization, and designing orthogonality-preserving gradient descent. For better scalability, we propose the stochastic OPT which performs orthogonal transformation stochastically for partial dimensions of neurons. Interestingly, OPT reveals that learning a proper coordinate system for neurons is crucial to generalization. We provide some insights on why OPT yields better generalization. Extensive experiments validate the superiority of OPT over the standard training.

1. Introduction

The inductive bias encoded in a neural network is generally determined by two major aspects: how the neural network is structured (*i.e.*, network architecture) and how the neural network is optimized (*i.e.*, training algorithm). For the same network architecture, using different training algorithms could lead to a dramatic difference in generalization performance [36, 60] even if the training loss is close to zero, implying that different training procedures lead to different inductive biases. Therefore, how to effectively train a neural network that generalize well remains an open challenge.

Recent theories [16, 15, 34, 45] suggest the importance of over-parameterization in linear neural networks. For example, [16] shows that optimizing an underdetermined quadratic objective over a matrix $\boldsymbol{M}$ with gradient descent on a factorization of $\boldsymbol{M}$ leads to an implicit regularization that may improve generalization. There is also strong empirical evidence [11, 51] that over-parameterzing the convolutional filters under some regularity is beneficial to generalization. Our paper aims to leverage the power of over-parameterization and explore more intrinsic structural priors in order to train a well-performing neural network.

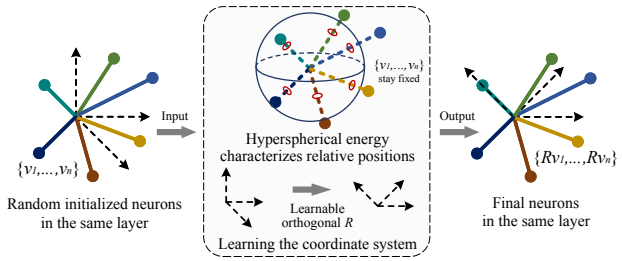

Figure 1: Overview of the orthogonal over-parameterized training framework. OPT learns an orthogonal transformation for each layer in the neural network, while keeping the randomly initialized neuron weights fixed.

Motivated by this goal, we propose a generic orthogonal over-parameterized training (OPT) framework for neural networks. Different from conventional neural training, OPT over-parameterizes a neuron $\boldsymbol{w}\in\mathbb{R}^d$ with the multiplication of a learnable layer-shared orthogonal matrix $\boldsymbol{R}\in\mathbb{R}^{d\times d}$ and a fixed randomly-initialized weight vector $\boldsymbol{v}\in\mathbb{R}^d$, and it follows that the equivalent weight for the neuron is $\boldsymbol{w}=\boldsymbol{R}\boldsymbol{v}$. Once each element of the neuron weight $\boldsymbol{v}$ has been randomly initialized by a zero-mean Gaussian distribution [20, 14], we fix them throughout the entire training process. Then OPT learns a layer-shared orthogonal transformation $\boldsymbol{R}$ that is applied to all the neurons (in the same layer). An illustration of OPT is given in Fig. 1. In contrast to standard neural training, OPT decomposes the neuron into an orthogonal transformation $\boldsymbol{R}$ that learns a proper coordinate system, and a weight vector $\boldsymbol{v}$ that controls the specific position of the neuron. Essentially, the weights $\{\boldsymbol{v}_1,\cdots,\boldsymbol{v}_n\in\mathbb{R}^d\}$ of different neurons determine the relative positions, while the layer-shared orthogonal matrix $\boldsymbol{R}$ specifies the coordinate system. Such a decoupled parameterization enables strong modeling flexibility.

Another motivation of OPT comes from an empirical observation that neural networks with lower *hyperspherical*

energy generalize better [49]. Hyperspherical energy quantifies the diversity of neurons on a hypersphere, and essentially characterizes the relative positions among neurons via this form of diversity. [49] introduces hyperspherical energy as a regularization in the network but do not guarantee that the hyperspherical energy can be effectively minimized (due to the existence of data fitting loss). To address this issue, we leverage the property of hyperspherical energy that it is independent of the coordinate system in which the neurons live and only depends on their relative positions. Specifically, we prove that, if we randomly initialize the neuron weight $\boldsymbol{v}$ with certain distributions, these neurons are guaranteed to attain minimum hyperspherical energy in expectation. It follows that OPT maintains the minimum energy during training by learning a coordinate system (*i.e.*, layer-shared orthogonal matrix) for the neurons. Therefore, OPT is able to provably minimize the hyperspherical energy.

We consider several ways to learn the orthogonal transformation. First, we unroll different orthogonalization algorithms such as Gram-Schmidt process, Householder reflection and Löwdin's symmetric orthogonalization. Different unrolled algorithms yield different implicit regularizations to construct the neuron weights. For example, symmetric orthogonalization guarantees that the new orthogonal basis has the least distance in the Hilbert space from the original non-orthogonal basis. Second, we consider to use a special parameterization (*e.g.*, Cayley parameterization) to construct the orthogonal matrix, which is more efficient in training. Third, we consider an orthogonality-preserving gradient descent to ensure that the matrix $\boldsymbol{R}$ stays orthogonal after each gradient update. Last, we relax the original optimization problem by making the orthogonality constraint a regularization for the matrix $\boldsymbol{R}$. Different ways of learning the orthogonal transformation may encode different inductive biases. We note that OPT aims to utilize orthogonalization as a tool to learn neurons that maintain small hyperspherical energy, rather than to study a specific orthogonalization method. Furthermore, we propose a refinement strategy to reduce the hyperspherical energy for the randomly initialized neuron weights $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$. In specific, we directly minimize the hyperspherical energy of these random weights as a preprocessing step before training them on actual data.

To improve scalability, we further propose the stochastic OPT that randomly samples neuron dimensions to perform orthogonal transformation. The random sampling process is repeated many times such that each dimension of the neuron is sufficiently learned. Finally, we provide some theoretical insights and discussions to justify the effectiveness of OPT. The advantages of OPT are summarized as follows:

- OPT is a generic neural network training framework with strong flexibility. There are many different ways to learn the orthogonal transformations and each one imposes a unique inductive bias. Our paper compares how different orthogonalizations may affect generalization in OPT.
- OPT is the first training framework where the hyperspherical energy is provably minimized (in contrast to [49]), leading to better empirical generalization. OPT reveals that learning a proper coordinate system is crucial to generalization, and the hyperspherical energy is sufficiently expressive to characterize relative neuron positions.
- There is no extra computational cost for the OPT-trained neural network in inference. In the testing stage, it has the same inference speed and model size as the normally trained network. Our experiments also show that OPT performs well on a diverse class of neural networks and therefore is agnostic to different neural architectures.
- Stochastic OPT can greatly improve the scalability of OPT while enjoying the same guarantee to minimize hyperspherical energy and having comparable performance.

2. Related Work

Orthogonality in Neural Networks. Orthogonality is widely adopted to improve neural networks. [4, 54, 7, 26, 78] use orthogonality as a regularization for neurons. [27, 42, 3, 75, 58, 31] use principled orthogonalization methods to guarantee the neurons are orthogonal to each other. In contrast to these works, OPT does not encourage orthogonality among neurons. Instead, OPT utilizes principled orthogonalization for learning orthogonal transformations for (not necessarily orthogonal) neurons to minimize hyperspherical energy.

Parameterization of Neurons. There are various ways to parameterize a neuron for different applications. [11] over-parameterizes a 2D convolution kernel by combining a 2D kernel of the same size and two additional 1D asymmetric kernels. The resulting convolution kernel has the same effective parameters during testing but more parameters during training. [51] constructs a neuron with a bilinear parameterization and regularizes the bilinear similarity matrix. [79] reparameterizes the neuron matrix with an adaptive fastfood transform to compress model parameters. [30, 48, 73] employ sparse and low-rank structures to construct convolution kernels for a efficient neural network.

Hyperspherical Learning. [54, 52, 72, 10, 71, 47, 50] propose to learn representations on a hypersphere and show that the angular information, in contrast to magnitude information, preserves the most semantic meaning. [49] define the hyperspherical energy that quantifies the diversity of neurons on a hypersphere and shows that the small hyperspherical energy generally improves empirical generalization.

3. Orthogonal Over-Parameterized Training

3.1. General Framework

OPT parameterizes the neuron as the multiplication of an orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ and a neuron weight vector $\boldsymbol{v} \in \mathbb{R}^d$, and the equivalent neuron weight becomes $\boldsymbol{w} = \boldsymbol{R}\boldsymbol{v}$. The output $\hat{y}$ of this neuron can be represented by $\hat{y} = (\boldsymbol{R}\boldsymbol{v})^\top \boldsymbol{x}$

where $\boldsymbol{x}\in\mathbb{R}^d$ is the input vector. In OPT, we typically fix the randomly initialized neuron weight $\boldsymbol{v}$ and only learn the orthogonal matrix $\boldsymbol{R}$. In contrast, the standard neuron is directly formulated as $\hat{y}=\boldsymbol{v}^\top\boldsymbol{x}$, where the weight vector $\boldsymbol{v}$ is learned via back-propagation in training.

As an illustrative example, we consider a linear MLP with a loss function $\mathcal{L}$ (*e.g.*, the least squares loss: $\mathcal{L}(e_1,e_2)=(e_1-e_2)^2$). Specifically, the learning objective of the standard training is $\min_{\{\boldsymbol{v}_i,u_i,\forall i\}}\sum_{j=1}^m\mathcal{L}\big(y,\sum_{i=1}^n u_i\boldsymbol{v}_i^\top\boldsymbol{x}_j\big)$, while differently, our OPT is formulated as

$$\min_{\{\boldsymbol{R},u_i,\forall i\}}\sum_{j=1}^m\mathcal{L}\big(y,\sum_{i=1}^n u_i(\boldsymbol{R}\boldsymbol{v}_i)^\top\boldsymbol{x}_j\big)\quad\text{s.t. }\boldsymbol{R}^\top\boldsymbol{R}=\boldsymbol{R}\boldsymbol{R}^\top=\boldsymbol{I}\quad(1)$$

where $\boldsymbol{v}_i\in\mathbb{R}^d$ is the i-th neuron in the first layer, and $\boldsymbol{u}=\{u_1,\cdots,u_n\}\in\mathbb{R}^n$ is the output neuron in the second layer. In OPT, each element of $\boldsymbol{v}_i$ is usually sampled from a zero-mean Gaussian distribution (*e.g.*, both Xavier [14] and Kaiming [20] initializations belong to this class), and is fixed throughout the entire training process. In general, OPT learns an orthogonal matrix that is applied to all the neurons instead of learning the individual neuron weight. Note that, we usually do not apply OPT to neurons in the output layer (*e.g.*, $\boldsymbol{u}$ in this MLP example, and the final linear classifiers in CNNs), since it makes little sense to fix a set of random linear classifiers. Therefore, the central problem is how to learn these layer-shared orthogonal matrices.

3.2. Hyperspherical Energy Perspective

One of the most important properties of OPT is its invariance to hyperspherical energy. Based on [49], the hyperspherical energy of n neurons is defined as $\boldsymbol{E}(\hat{\boldsymbol{v}}_i|_{i=1}^n)=\sum_{i=1}^n\sum_{j=1,j\neq i}^n\|\hat{\boldsymbol{v}}_i-\hat{\boldsymbol{v}}_j\|^{-1}$ in which $\hat{\boldsymbol{v}}_i=\frac{\boldsymbol{v}_i}{\|\boldsymbol{v}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere $\mathbb{S}^{d-1}=\{\boldsymbol{v}\in\mathbb{R}^d|\,\|\boldsymbol{v}\|=1\}$. Hyperspherical energy is used to characterize the diversity of n neurons on a unit hypersphere. Assume that we have n neurons in one layer, and we have learned an orthogonal matrix $\boldsymbol{R}$ for these neurons. The hyperspherical energy of these n OPT-trained neurons is

$$\begin{aligned}\boldsymbol{E}(\hat{\boldsymbol{R}}\hat{\boldsymbol{v}}_i|_{i=1}^n)&=\sum_{i=1}^n\sum_{j=1,j\neq i}^n\|\boldsymbol{R}\hat{\boldsymbol{v}}_i-\boldsymbol{R}\hat{\boldsymbol{v}}_j\|^{-1}\\(\text{since }\|\boldsymbol{R}\|^{-1}=1)\quad&=\sum_{i=1}^n\sum_{j=1,j\neq i}^n\|\hat{\boldsymbol{v}}_i-\hat{\boldsymbol{v}}_j\|^{-1}=\boldsymbol{E}(\hat{\boldsymbol{v}}_i|_{i=1}^n)\end{aligned}\quad(2)$$

which verifies that the hyperspherical energy does not change in OPT. Moreover, [49] proves that minimum hyperspherical energy corresponds to the uniform distribution over the hypersphere. As a result, if the initialization of the neurons in the same layer follows the uniform distribution over the hypersphere, then we can guarantee that the hyperspherical energy is minimal in a probabilistic sense.

Theorem 1. *For the neuron $\boldsymbol{h}=\{h_1,\cdots,h_d\}$ where $h_i,\forall i$ are initialized i.i.d. following a zero-mean Gaussian distribution (i.e., $h_i\sim N(0,\sigma^2)$), the projections onto a unit hypersphere $\hat{\boldsymbol{h}}=\boldsymbol{h}/\|\boldsymbol{h}\|$ where $\|\boldsymbol{h}\|=(\sum_{i=1}^d h_i^2)^{1/2}$ are uniformly distributed on the unit hypersphere $\mathbb{S}^{d-1}$. The neurons with minimum hyperspherical energy attained asymptotically approach the uniform distribution on $\mathbb{S}^{d-1}$.*

Theorem 1 proves that, as long as we initialize the neurons in the same layer with zero-mean Gaussian distribution, the resulting hyperspherical energy is guaranteed to be small (*i.e.*, the expected energy is minimal). It is because the neurons are uniformly distributed on the unit hypersphere and hyperspherical energy quantifies the uniformity on the hypersphere in some sense. More importantly, prevailing neuron initializations such as [14] and [20] are zero-mean Gaussian distribution. Therefore, our neurons naturally have low hyperspherical energy from the beginning. Appendix L gives geometric properties of the random initialized neurons.

3.3. Unrolling Orthogonalization Algorithms

In order to learn the orthogonal transformation, we unroll classic orthogonalization algorithms and embed them into the neural network such that the training can be performed in an end-to-end fashion. We need to make every step of the orthogonalization algorithm differentiable, as shown in Fig. 2.

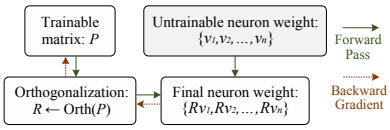

Figure 2: Unrolled orthogonalization.

Gram-Schmidt Process. This method takes a linearly independent set and eventually produces an orthogonal set based on it. The Gram-Schmidt Process (GS) usually takes the following steps to orthogonalize a set of vectors $\{\boldsymbol{u}_1,\cdots,\boldsymbol{u}_n\}\in\mathbb{R}^{n\times n}$ and obtain an orthonormal set $\{\boldsymbol{e}_1,\cdots,\boldsymbol{e}_i,\cdots,\boldsymbol{e}_n\}\in\mathbb{R}^{n\times n}$. First, when $i=1$, we have $\boldsymbol{e}_1=\frac{\tilde{\boldsymbol{e}}_1}{\|\tilde{\boldsymbol{e}}_1\|}$ where $\tilde{\boldsymbol{e}}_1=\boldsymbol{u}_1$. Then, when $n\geq i\geq 2$, we have $\boldsymbol{e}_i=\frac{\tilde{\boldsymbol{e}}_i}{\|\tilde{\boldsymbol{e}}_i\|}$ where $\tilde{\boldsymbol{e}}_i=\boldsymbol{u}_i-\sum_{j=1}^{i-1}\text{Proj}_{\boldsymbol{e}_j}(\boldsymbol{u}_i)$. Note that, $\text{Proj}_{\boldsymbol{b}}(\boldsymbol{a})=\frac{\langle\boldsymbol{a},\boldsymbol{b}\rangle}{\langle\boldsymbol{b},\boldsymbol{b}\rangle}\boldsymbol{b}$ is defined as the projection operator.

Householder Reflection. A Householder reflector is defined as $\boldsymbol{H}=\boldsymbol{I}-2\frac{\boldsymbol{u}\boldsymbol{u}^\top}{\|\boldsymbol{u}\|^2}$ where $\boldsymbol{u}$ is perpendicular to the reflection hyperplane. In QR factorization, Householder reflection (HR) is used to transform a (non-singular) square matrix into an orthogonal matrix and an upper triangular matrix. Given a matrix $\boldsymbol{U}=\{\boldsymbol{u}_1,\cdots,\boldsymbol{u}_n\}\in\mathbb{R}^{n\times n}$, we consider the first column vector $\boldsymbol{u}_1$. We use Householder reflector to transform $\boldsymbol{u}_1$ to $\boldsymbol{e}_1=\{1,0,\cdots,0\}$. Specifically, we construct an orthogonal matrix $\boldsymbol{H}_1$ with $\boldsymbol{H}_1=\boldsymbol{I}-2\frac{(\boldsymbol{u}_1-\|\boldsymbol{u}_1\|\boldsymbol{e}_1)(\boldsymbol{u}_1-\|\boldsymbol{u}_1\|\boldsymbol{e}_1)^\top}{\|\boldsymbol{u}_1-\|\boldsymbol{u}_1\|\boldsymbol{e}_1\|^2}$. The first column of $\boldsymbol{H}_1\boldsymbol{U}$ becomes $\{\|\boldsymbol{u}_1\|,0,\cdots,0\}$. At the k-th step, we can view the sub-matrix $\boldsymbol{U}_{(k:n,k:n)}$ as a new $\boldsymbol{U}$, and use the same procedure to construct the Householder transformation $\tilde{\boldsymbol{H}}_k\in\mathbb{R}^{(n-k)\times(n-k)}$. We construct the final Householder transformation as $\boldsymbol{H}_k=\text{Diag}(\boldsymbol{I}_k,\tilde{\boldsymbol{H}}_k)$. Now we can gradually transform $\boldsymbol{U}$ to an upper triangular matrix with n Householder reflections. Therefore, we have that $\boldsymbol{H}_n\cdots\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{U}=\boldsymbol{R}^{\text{up}}$ where $\boldsymbol{R}^{\text{up}}$ is an upper triangular matrix and the obtained

orthogonal set is $\boldsymbol{Q}^\top = \boldsymbol{H}_n \cdots \boldsymbol{H}_2 \boldsymbol{H}_1$.

Löwdin's Symmetric Orthogonalization. Let the matrix $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\} \in \mathbb{R}^{n \times n}$ be a given set of linearly independent vectors in an n-dimensional space. A non-singular linear transformation $\boldsymbol{A}$ can transform the basis $\boldsymbol{U}$ to an orthogonal basis $\boldsymbol{R}$: $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{A}$. The matrix $\boldsymbol{R}$ will be orthogonal if $\boldsymbol{R}^\top \boldsymbol{R} = (\boldsymbol{U}\boldsymbol{A})^\top \boldsymbol{U}\boldsymbol{A} = \boldsymbol{A}^\top \boldsymbol{M}\boldsymbol{A} = \boldsymbol{I}$ where $\boldsymbol{M} = \boldsymbol{U}^\top \boldsymbol{U}$ is the Gram matrix of the given set $\boldsymbol{U}$. We obtain a general solution to the orthogonalization problem via the substitution: $\boldsymbol{A} = \boldsymbol{M}^{-\frac{1}{2}} \boldsymbol{B}$ where $\boldsymbol{B}$ is an arbitrary unitary matrix. The specific choice $\boldsymbol{B} = \boldsymbol{I}$ gives the Löwdin's symmetric orthogonalization (LS): $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{M}^{-\frac{1}{2}}$. We can analytically obtain the symmetric orthogonalization from the singular value decomposition: $\boldsymbol{U} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Then LS gives $\boldsymbol{R} = \boldsymbol{W}\boldsymbol{V}^\top$ as the orthogonal set for $\boldsymbol{U}$. LS has a unique property which the other orthogonalizations do not have. The orthogonal set resembles the original set in a nearest-neighbour sense. More specifically, LS guarantees that $\sum_i \|\boldsymbol{R}_i - \boldsymbol{U}_i\|^2$ (where $\boldsymbol{R}_i$ and $\boldsymbol{U}_i$ are the i-th column of $\boldsymbol{R}$ and $\boldsymbol{U}$, respectively) is minimized. Intuitively, LS indicates the gentlest pushing of the directions of the vectors in order to get them orthogonal to each other.

Discussion. These orthogonalization algorithms are fully differentiable and end-to-end trainable. For accurate orthogonality, these algorithms can be used repeatedly and unrolled with multiple steps. Empirically, one-step unrolling already works well. Givens rotations can also construct the orthogonal matrix, but it requires traversing all lower triangular elements in the original set $\boldsymbol{U}$, which takes $\mathcal{O}(n^2)$ complexity and is too costly. Interestingly, each orthogonalization encodes a unique inductive bias to the neurons by imposing implicit regularizations (*e.g.*, least distance in Hilbert space for LS). Details about these orthogonalizations are in Appendix A. Unrolling orthogonalization has been considered in different scenarios [27, 69, 56]. More orthogonalization methods [41] can be applied in OPT, but exhaustively applying them to OPT is out of the scope of this paper.

3.4. Orthogonal Parameterization

A convenient way to ensure orthogonality while learning the matrix $\boldsymbol{R}$ is to use a special parameterization that inherently guarantees orthogonality. The exponential parameterization use $\boldsymbol{R} = \exp(\boldsymbol{W})$ (where $\exp(\cdot)$ denotes the matrix exponential) to represent an orthogonal matrix from a skew-symmetric matrix $\boldsymbol{W}$. The Cayley parameterization (CP) is a Padé approximation of the exponential parameterization, and is a more natural choice due to its simplicity. CP uses the following transform to construct an orthogonal matrix $\boldsymbol{R}$ from a skew-symmetric matrix $\boldsymbol{W}$: $\boldsymbol{R} = (\boldsymbol{I} + \boldsymbol{W})(\boldsymbol{I} - \boldsymbol{W})^{-1}$ where $\boldsymbol{W} = -\boldsymbol{W}^\top$. We note that CP only produces the orthogonal matrices with determinant 1, which belong to the special orthogonal group and thus $\boldsymbol{R} \in SO(n)$. Specifically, it suffices to learn the upper or lower triangular of the matrix $\boldsymbol{W}$ with unconstrained optimization to obtain a desired orthogonal matrix $\boldsymbol{R}$. Cayley parameterization does not cover the entire orthogonal group and is less flexible in terms of representation power, which serves as an explicit regularization for the neurons.

3.5. Orthogonality-Preserving Gradient Descent

An alternative way to guarantee orthogonality is to modify the gradient update for the matrix $\boldsymbol{R}$. The idea is to initialize $\boldsymbol{R}$ with an arbitrary orthogonal matrix and then ensure each gradient update is to apply an orthogonal transformation to $\boldsymbol{R}$. It is essentially conducting gradient descent on the Stiefel manifold [44, 74, 75, 42, 3, 22, 33]. Given a matrix $\boldsymbol{U}_{(0)} \in \mathbb{R}^{n \times n}$ that is initialized as an orthogonal matrix, we aim to construct an orthogonal transformation as the gradient update. We use the Cayley transform to compute a parametric curve on the Stiefel manifold $\mathcal{M}_s = \{\boldsymbol{U} \in \mathbb{R}^{n \times n} : \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}\}$ with a specific metric via a skew-symmetric matrix $\boldsymbol{W}$ and use it as the update rule:

$$\boldsymbol{Y}(\lambda) = (\boldsymbol{I} - \frac{\lambda}{2}\boldsymbol{W})^{-1}(\boldsymbol{I} + \frac{\lambda}{2}\boldsymbol{W})\boldsymbol{U}_{(i)},\ \boldsymbol{U}_{(i+1)} = \boldsymbol{Y}(\lambda) \quad (3)$$

where $\hat{\boldsymbol{W}} = \nabla f(\boldsymbol{U}_{(i)})\boldsymbol{U}_{(i)}^\top - \frac{1}{2}\boldsymbol{U}_{(i)}(\boldsymbol{U}_{(i)}^\top \nabla f(\boldsymbol{U}_{(i)}\boldsymbol{U}_{(i)}^\top)$ and $\boldsymbol{W} = \hat{\boldsymbol{W}} - \hat{\boldsymbol{W}}^\top$. $\boldsymbol{U}_{(i)}$ denotes the orthogonal matrix in the i-th iteration. $\nabla f(\boldsymbol{U}_{(i)})$ denotes the original gradient of the loss function *w.r.t.* $\boldsymbol{U}_{(i)}$. We term this gradient update as orthogonal-preserving gradient descent (OGD). To reduce the computational cost of the matrix inverse in Eq. 3, we use an iterative method [44] to approximate the Cayley transform without matrix inverse. We arrive at the fixed-point iteration:

$$\boldsymbol{Y}(\lambda) = \boldsymbol{U}_{(i)} + \frac{\lambda}{2}\boldsymbol{W}\left(\boldsymbol{U}_{(i)} + \boldsymbol{Y}(\lambda)\right) \quad (4)$$

which converges to the closed-form Cayley transform with a rate of $o(\lambda^{2+n})$ (n is the iteration number). In practice, two iterations suffice for a reasonable approximation accuracy.

3.6. Relaxation to Orthogonal Regularization

Alternatively, we also consider relaxing the original optimization with an orthogonality constraint to an unconstrained optimization with orthogonality regularization (OR). Specifically, we remove the orthogonality constraint, and adopt an orthogonality regularization for $\boldsymbol{R}$, *i.e.*, $\|\boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{I}\|_F^2$. However, OR cannot guarantee the energy stays unchanged. Taking Eq. 1 as an example, the objective becomes

$$\min_{\boldsymbol{R}, u_i, \forall i} \sum_{j=1}^{m} \mathcal{L}\big(y, \sum_{i=1}^{n} u_i (\boldsymbol{R}\boldsymbol{v}_i)^\top \boldsymbol{x}_j\big) + \beta \|\boldsymbol{R}^\top \boldsymbol{R} - \boldsymbol{I}\|_F^2 \quad (5)$$

where β is a hyperparameter. This serves as an relaxation of the original OPT objective. Note that, OR is imposed to $\boldsymbol{R}$ instead of neurons and is quite different from the existing orthogonality regularization on neurons [54, 4, 27, 78, 7].

3.7. Refining the Initialization as Preprocessing

Minimizing the energy beforehand. Because we randomly initialize the neurons $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$, there exists a variance that makes the hyperspherical energy deviate from

the minima even if the hyperspherical energy is minimal in a probabilistic sense. To further reduce the hyperspherical energy, we propose to refine the random initialization by minimizing its hyperspherical energy as a preprocessing step before the OPT training. Specifically, before feeding these neurons to OPT, we first minimize the hyperspherical energy of the initialized neurons with gradient descent (without fitting the training data). Moreover, since the randomly initialized neurons cannot guarantee to get rid of the collinearity redundancy as shown in [49] (*i.e.*, two neurons are on the same line but have opposite directions), we can perform the half-space hyperspherical energy minimization [49].

Normalizing the neurons. The norm of the randomly initialized neurons may have some influence on OPT, serving a role similar to weighting the importance of different neurons. Moreover, the norm makes the hyperspherical energy less expressive to characterize the diversity of neurons, as discussed in Section 5.3. To make the coordinate frame (i.e. the rotation matrix R) truly independent of the relative positions of the neurons, we propose to normalize the neuron weights such that each neuron has unit norm. Because the weights of the neurons $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$ are fixed during training and orthogonal matrices will not change the norm of the neurons, we only need to normalize the randomly initialized neuron weights as a preprocessing before the OPT training.

We have comprehensively evaluated both refinement strategies in Section 6.2 and verified their effectiveness. Note that the effectiveness of OPT is not dependent on these refinements. Our experiments do not use these refinements by default and the results show that OPT still performs well.

4. Towards Better Scalablity for OPT

If the dimension of neurons becomes extremely large, then the orthogonal matrix to transform the neurons will also be large. Therefore, it may take large GPU memory and time to train the neural networks with the original OPT. To address this, we propose a scalable variant – stochastic OPT (S-OPT). The key idea of S-OPT is to randomly select some dimensions from the neurons in the same layer and construct a small orthogonal matrix to transform these dimensions together. The selection of dimensions is stochastic in each outer iteration, so a small orthogonal matrix is sufficient to cover all the neuron dimensions. S-OPT aims to approximate a large orthogonal transformation for all the neuron dimensions with many small orthogonal transformations for random subsets of these dimensions, which shares similar spirits with Givens rotation. The approximation will be more accurate when the procedure is randomized over many times. Fig. 3 compares the size of the orthogonal matrix in OPT and S-OPT. The orthogonal matrix in OPT is of size $d \times d$, while the orthogonal matrix in S-OPT is of size $p \times p$ where p is usually much smaller than d. Most importantly, S-OPT can still preserve the low hyperspherical energy of neurons because of the following result.

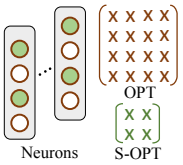

Figure 3: S-OPT.

Theorem 2. *For n d-dimensional neurons, selecting any p ($p \leq d$) dimensions and applying an shared orthogonal transformation ($p \times p$ orthogonal matrix) to these p dimensions of all neurons will not change the hyperspherical energy.*

A description of S-OPT is given in Algorithm 1. S-OPT has outer and inner iterations. In each inner iteration, the training is almost the same as OPT, except that the orthogonal matrix transforms a subset of the dimensions and the learnable orthogonal matrix has to be re-initialized to an identity matrix. The selection of neuron dimension is randomized in every outer iteration such that all neuron dimensions can be sufficiently covered as the number of outer iterations increases. Therefore, given sufficient number of iterations, S-OPT will perform comparably to OPT, as empirically verified in Section 6.3. As a parallel direction to improve the scalability, we further propose a parameter-efficient OPT in Appendix I. This OPT variant explores structure priors in $\boldsymbol{R}$ to improve parameter efficiency.

Algorithm 1 Stochastic OPT

for $i = 1, 2, \cdots, N_{\text{out}}$ **do**
  for $j = 1, 2, \cdots, N_{\text{in}}$ **do**
    1. Randomly select p dimensions from d-dimensional neurons in the same layer.
    2. Construct an orthogonal matrix $\boldsymbol{R}_p \in \mathbb{R}^{p \times p}$ and initialize it as identity matrix.
    3. Update $\boldsymbol{R}_p$ by applying OPT with one iteration.
  end
  4. Multiply $\boldsymbol{R}_p$ back to the p-dim sub-vectors from the d-dim neurons to transform these neurons.
end

5. Intriguing Insights and Discussions

5.1. Local Landscape

We follow [43] to visualize the loss landscapes of both standard training and OPT in Fig. 4. For standard training, we perturb the parameter space of all the neurons (*i.e.*, filters). For OPT, we perturb the parameter space of all the trainable matrices (*i.e.*, $\boldsymbol{P}$ in Fig. 2), because OPT does not directly learn neuron weights. The general idea is to use two random vectors (*e.g.*, normal distribution) to perturb the parameter space and obtain the loss value with the perturbed network parameters. Details and full results about the visualization are given in Appendix E. The loss landscape of standard training has extremely sharp minima. The red region is very flat, leading to small gradients. In contrast, the loss landscape of OPT is much more smooth and convex with flatter minima, well matching the finding that flat minimizers generalize well [23, 8, 29]. Additional loss landscape visualization results in Appendix F (with uniform perturbation distributions) also support the same argument.

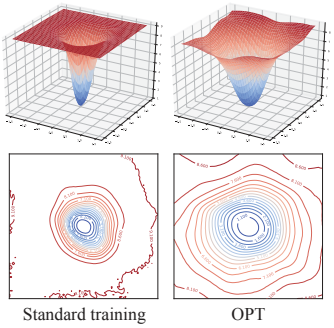

Figure 4: Training loss landscapes.

We also show the landscape of testing error on CIFAR-100 in Fig. 5. Full results and details are in Appendix E. Compared to standard training, the testing error of OPT increases more slowly and smoothly while the network parameters move away from the minima, which indicates that the parameter space of OPT yields better robustness to perturbations.

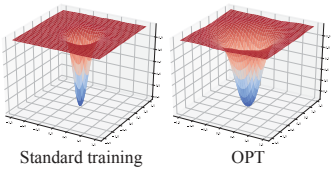

Figure 5: Testing error landscapes.

5.2. Optimization and Generalization

We discuss why OPT may improve optimization and generalization. On one hand, [77] proves that once the neurons are hyperspherically diverse enough in a one-hidden-layer network, the training loss is on the order of the square norm of the gradient and the generalization error will have an additional term $\tilde{\mathcal{O}}(1/\sqrt{m})$ where m is the number of samples. This suggests that SGD-optimized networks with minimum hyperspherical energy (MHE) attained have no spurious local minima. Since OPT is guaranteed to achieve MHE in expectation, OPT-trained networks enjoy the inductive bias induced by MHE. On the other hand, [34, 1, 12, 45, 16] shows that over-parameterization in neural networks improves the first-order optimization, leads to better generalization, and imposes implicit regularizations. In the light of this, OPT also introduces over-parameterization to each neuron, which shares similar spirits with [46]. Specifically, one d-dimensional neuron has d^2+d parameters in OPT (with d^2 being layer-shared), compared to d parameters in a standard neuron. Although OPT uses more parameters for a neuron in training, the equivalent number of parameters for a neuron stays unchanged and it will not affect testing speed.

5.3. Discussions

Over-parameterization. We delve deeper into the over-parameterization in the context of OPT. Its definition varies in different cases. OPT is over-parameterized in terms of training in the following sense. Although OPT-trained networks have the same effective number of parameters as the standard networks in testing, the OPT neuron is decomposed into two sets of parameters in training: orthogonal matrix and neuron weights. It means that the same set of parameters in a neural network can be represented by different sets of training parameters in OPT (*i.e.*, different combinations of orthogonal matrices and neuron weights can lead to the same neural network). OPT is still over-parameterized even if we only count the number of learnable parameters. For a layer of n d-dimensional neurons, the number of learnable parameters in vanilla OPT is $\frac{d(d-1)}{2}$ in contrast to nd in standard training. In prevailing architectures (*e.g.*, ResNet [21]), the neuron dimension is far larger than the number of neurons.

Coordinate system and relative position. OPT shows that learning the coordinate system yields better generalization than learning neuron weights directly. This implies that the coordinate system is crucial to generalization. However, the relative position does not matter only when the hyperspherical energy is sufficiently low, indicating that the neurons need to be diverse enough on the unit hypersphere.

The effects of neuron norm. Because we will normalize the neuron norm when computing the hyperspherical energy, the effects of neuron norm will not be taken into consideration. Moreover, simply learning the orthogonal matrices will not change the neuron norm. Therefore, the neuron norm may affect the training. We use an extreme example to demonstrate the effects. Assume that one of the neurons has norm 1000 and the other neurons have norm 0.01. Then no matter what orthogonal matrices we have learned, the final performance will be bad. In this case, the hyperspherical energy can still be minimized to a very low value, but it can not capture the norm distribution. Fortunately, such an extreme case is unlikely to happen, because we are using zero-mean Gaussian distribution to initialize the neuron and every neuron also has the same expected value for the norm. To eliminate the effects of norms, we can normalize the neuron weights in training, as proposed in Section 3.7.

6. Applications and Experimental Results

We put all the experimental settings and many additional results in Appendix D and Appendix I,K, respectively.

6.1. Ablation Study and Exploratory Experiments

Orthogonality. We evaluate whether orthogonality in OPT is necessary. We use 6-layer and 9-layer CNN (Appendix D) on CIFAR-100. Then we compare OPT with unconstrained over-parameterized training (UPT) which learns an unconstrained matrix $\boldsymbol{R}$ (with weight decay) using the same network. In Table 1, "FN" denotes whether the randomly initialized neuron weights are fixed in training. "LR" denotes whether the learnable matrix $\boldsymbol{R}$ is unconstrained ("U") or orthogonal ("GS" for Gram-Schmidt process). Table 1 shows that without orthogonality, UPT performs much worse than OPT.

Method	FN	LR	CNN-6	CNN-9
Baseline	-	-	37.59	33.55
UPT	✗	U	48.47	46.72
UPT	✓	U	42.61	39.38
OPT	✗	GS	37.24	32.95
OPT	✓	GS	**33.02**	**31.03**

Table 1: Error (%) on C-100.

Fixed or learnable weights. From Table 1, we can see that using fixed neuron weights is consistently better than learnable neuron weights in both UPT and OPT. It indicates that fixing the neuron weights can well maintain low hyperspherical energy and is beneficial to empirical generalization.

Refining initialization. We evaluate two refinement methods in Section 3.7 for neuron initialization. First, we consider the hyperspherical energy minimization as a preprocessing for the neuron weights. Our experiment uses CNN-6 on CIFAR-100. Specifically, we run gradient descent for 5k

Method	Original	MHE	HS-MHE	CoMHE
OPT (GS)	33.02	32.99	32.78	**32.69**
OPT (LS)	34.48	34.43	34.37	**34.15**
OPT (CP)	33.53	33.50	33.42	**33.27**
Energy	3.5109	3.5003	3.4976	**3.4954**

Table 2: Refining initialized energy.

iterations to minimize the objective of MHE/HS-MHE [49] or CoMHE [47] before the training starts. Table 2 shows the hyperspherical energy before and after the preprocessing. All methods start with the same random initialization, so all hyperspherical energies start at 3.5109. Testing errors (%) in Table 2 show that the refinement well improves OPT. Although using advanced regularizations such as CoMHE as pre-processing can improve the performance significantly, we do not use them in the other experiments in order to keep our comparison fair and clean. More different ways to minimize the hyperspherical energy can also be considered [50].

We evaluate the second refinement strategy, *i.e.*, neuron weight normalization. Section 5.3 has explained why normalizing the neuron weights may be useful. After initialization, we normalize all the neuron weights to 1. Since OPT does not change the neuron norm, the neuron will keep the norm as 1. More importantly, the hyperspherical energy will not be affected by the neuron normalization. We conduct classification with CNN-6 on CIFAR-100. Testing errors in Table 3 show that normalizing the neurons greatly improves OPT, validating our previous analysis. Note that, these two refinements are not used by default in other experiments.

Method	w/o Norm	w/ Norm
Baseline	37.59	**36.05**
OPT (GS)	33.02	**32.54**
OPT (HR)	35.67	**35.30**
OPT (LS)	34.48	**32.11**
OPT (CP)	33.53	**32.49**
OPT (OGD)	33.37	**32.70**
OPT (OR)	34.70	**33.27**

Table 3: Normalization (%).

High vs. low energy. We validate that high hyperspherical energy corresponds to inferior empirical generalization. To initialize high energy neurons, we use [20] and set the mean as 1e-3, 1e-2, 2e-2, and 3e-2. We experiment on CIFAR-100 with CNN-6. Table 4 ("N/C" denotes not converged) show that higher energy generalizes worse and also leads to difficulty in convergence. We see that a small change in energy can lead to a dramatic generalization gap.

Mean	Energy	Error (%)
0	**3.5109**	**32.49**
1e-3	3.5117	33.11
1e-2	3.5160	39.51
2e-2	3.5531	53.89
3e-2	3.6761	N/C

Table 4: Initial energy.

No BatchNorm. We evaluate how OPT performs without BatchNorm (BN) [28]. We perform classification on CIFAR-100 with CNN-6. In Table 5, we see that all OPT variants consistently outperform both the baseline and HS-MHE [49] by a significant margin, validating that OPT can work well without BN. CP achieves the best error with more than 4% lower than standard training.

Method	Error (%)
Baseline	38.95
HS-MHE	36.90
OPT (GS)	35.61
OPT (HR)	37.51
OPT (LS)	35.83
OPT (CP)	**34.88**
OPT (OGD)	35.38

Table 5: No BN.

6.2. Empirical Evaluation on OPT

Multi-layer perceptrons. We evaluate OPT on MNIST with a 3-layer MLP. Appendix D gives specific settings. Table 6 shows the testing error with normal initialization (MLP-N) or Xavier initialization [14] (MLP-X). GS/HR/LS denote different orthogonalization unrolling. CP denotes Cayley parameterization. OGD denotes orthogonal-preserving gradient descent. OR denotes relaxed orthogonal regularization. All OPT variants outperform the others by a large margin.

Method	MNIST		CIFAR-100			
	MLP-N	MLP-X	CNN-6	CNN-9	ResNet-20	ResNet-32
Baseline	6.05	2.14	37.59	33.55	31.11	30.16
Orthogonal [7]	5.78	1.93	36.32	33.24	31.06	30.05
SRIP [4]	-	-	34.82	32.72	30.89	29.70
HS-MHE [49]	5.57	1.88	34.97	32.87	30.98	29.76
OPT (GS)	**5.11**	**1.45**	**33.02**	**31.03**	30.49	29.34
OPT (HR)	5.31	1.60	35.67	32.75	30.73	29.56
OPT (LS)	5.32	1.54	34.48	31.22	30.51	29.42
OPT (CP)	5.14	1.49	33.53	31.28	**30.47**	**29.31**
OPT (OGD)	5.38	1.56	33.33	31.47	30.50	29.39
OPT (OR)	5.41	1.78	34.70	32.63	30.66	29.47

Table 6: Testing error (%) of OPT for MLPs and CNNs.

Convolutional networks. We evaluate OPT with 6/9-layer plain CNNs and ResNet-20/32 [21] on CIFAR-100. Detailed settings are in Appendix D. All neurons (*i.e.*, convolution kernels) are initialized by [20]. BatchNorm is used by default. Table 6 shows that all OPT variants outperform both baseline and HS-MHE by a large margin. HS-MHE puts the hyperspherical energy into the loss function and naively minimizes it along with the CNN. We observe that OPT (HR) performs the worse among all OPT variants partially because of its intensive unrolling computation. OPT (GS) achieves the best testing error on CNN-6/9, while OPT (CP) achieves the best testing error on ResNet-20/34, implying that different OPT encodes different inductive bias.

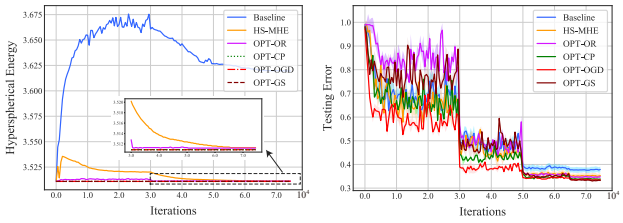

Figure 6: Training dynamics on CIFAR-100. Left: Hyperspherical energy vs. iteration. Right: Testing error vs. iteration.

Training dynamics. We look into how hyperspherical energy and testing error changes in OPT. Fig. 6 shows that the energy of the baseline will increase dramatically at the beginning and then gradually go down, but it still stays in a high value in the end. HS-MHE well reduces the energy at the end of the training. In contrast, OPT variants always maintain very small energy in training. OPT with GS, CP and OGD keep exactly the same energy as the random initialization, while OPT (OR) slightly increases the energy due to relaxation. All OPT variants converge efficiently and stably.

Large-scale learning. To see how OPT performs in large-scale settings, we evaluate OPT on the large-scale ImageNet-2012 [62]. Specifically, we use OPT with OGD and CP to train a plain 10-layer CNN (Appendix D) on ImageNet. Note that, our purpose is to validate the superiority of OPT over the corresponding baseline rather than achieving state-of-the-art results. Table 7 shows that OPT (CP) reduces top-1 and top-5 error for the baseline by $\sim$0.7% and $\sim$0.9%, respectively.

Method	Top-1	Top-5
Baseline	44.32	21.13
Orthogonal [7]	44.13	20.97
HS-MHE [49]	43.92	20.85
OPT (OGD)	43.81	20.49
OPT (CP)	**43.67**	**20.26**

Table 7: ImageNet (%).

Few-shot recognition. For evaluating OPT on cross-task generalization, we perform the few-shot recognition on Mini-ImageNet, following the same setup as [9]. Appendix D gives more detailed settings. We apply OPT with CP to train the baseline and baseline++ in [9], and immediately obtain improvements. Therefore, OPT-trained networks generalize well in this challenging scenario.

Method	5-shot Acc. (%)
MAML [13]	62.71 ± 0.71
MatchingNet [70]	63.48 ± 0.66
ProtoNet [65]	64.24 ± 0.72
Baseline [9]	62.53 ± 0.69
Baseline w/ OPT	**63.27 ± 0.68**
Baseline++ [9]	66.43 ±0.63
Baseline++ w/ OPT	**66.82 ± 0.62**

Table 8: Few-shot learning.

Geometric learning. We apply OPT to graph convolution network (GCN) [37] and point cloud network (PointNet) [57] for graph node and point cloud classification, respectively. The training of GCN and PointNet is conceptually similar to MLP, and the detailed training procedures are given in Appendix D. For GCN, we evaluate OPT on Cora and Pubmed datsets [63]. For PointNet, we conduct experiments on ModelNet-40 dataset [76]. Table 9 shows that OPT effectively improves both GCN and PointNet.

Method	GCN		PointNet
	Cora	Pubmed	MN-40
Baseline	81.3	79.0	87.1
OPT (GS)	81.9	79.4	87.23
OPT (CP)	82.0	79.4	87.81
OPT (OGD)	**82.3**	**79.5**	**87.86**

Table 9: Geometric networks.

6.3. Empirical Evaluation on S-OPT

Method	CIFAR-100				ImageNet	
	CNN-6	Params	Wide CNN-9	Params	ResNet-18	Params
Baseline	37.59	258K	28.03	2.99M	32.95	11.7M
HS-MHE [49]	34.97	258K	25.96	2.99M	32.50	11.7M
OPT (GS)	**33.02**	1.36M	OOM	16.2M	OOM	46.5M
S-OPT (GS)	33.70	**90.9K**	**25.59**	**1.04M**	**32.26**	**3.39M**

Table 10: OPT vs. S-OPT on CIFAR-100 & ImageNet.

Convolutional networks. S-OPT is a scalable OPT variant, and we evaluate its performance in terms of number of *trainable parameters* and testing error. Training parameters are learnable variables in training, and are different from model parameters in testing. In testing, all methods have the same number of model parameters. We perform classification on CIFAR-100 with CNN-6 and wide CNN-9. We also evaluate S-OPT with standard ResNet-18 on ImageNet. Detailed settings are in Appendix D. For S-OPT, we set the sampling dimension as 25% of the original neuron dimension in each layer. Table 10 shows that S-OPT achieves a good trade-off between accuracy and scalability. More importantly, S-OPT can be applied to large neural networks, making OPT more useful in practice. Additionally, Appendix I discusses an efficient parameter sharing for OPT.

Sampling dimensions. We study how the sampling dimension p affect the performance by performing classification with wide CNN-9 on CIFAR-100. In Table 11, $p=d/4$ means that we randomly sample $1/4$ of the original neuron dimension in each layer, so p may vary in different layer. $p=16$ means that we sample 16 dimensions in each layer. Note that there are 25.6K parameters used for the final classification layer, which can not be saved in S-OPT. Table 11 shows that S-OPT can achieve highly competitive accuracy with a reasonably large p.

$p=$	Error (%)	Params
d	OOM	16.2M
$d/4$	**25.59**	1.04M
$d/8$	28.61	278K
$d/16$	32.52	88.7K
16	33.03	27.0K
3	45.22	26.0K
0	60.64	**25.6K**

Table 11: Sampling dim.

6.4. Large Categorical Training

Previously, OPT is not applied to the final classification layer, since it makes little sense to fix random classifiers and learn an orthogonal matrix to transform them. However, learning the classification layer can be costly with large number of classes. The number of trainable parameters of the classification layer grows linearly with the number of classes. To address this, OPT can be used to learn the classification layer, because its number of trainable parameters only depends on the classifier dimension. To be fair, we *only* learn the last classification layer with OPT and the other layers are normally learned (CLS-OPT). The oracle learns the entire network normally. Experimental details are in Appendix D.

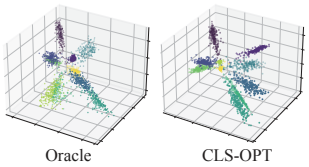

Figure 7: Feature visualization.

We intuitively compare the oracle and CLS-OPT by visualizing the deep MNIST features following [53]. The features are the direct outputs of CNN by setting the output dimension as 3. Fig. 7 show that even if CLS-OPT fixes randomly initialized classifiers, it can still learn discriminative and separable deep features.

Method	ResNet-18A		ResNet-18B	
	Error	Params	Error	Params
Oracle	**18.08**	64.0K	12.12	512K
CLS-OPT	21.12	**8.13K**	**12.05**	**131K**

Table 12: CLS-OPT on ImageNet.

We evaluate its performance on ImageNet with 1K classes. We use ResNet-18 with different output dimensions (A:128, B:512). Table 12 gives the top-5 test error (%) and "Params" denotes the number of trainable parameters in the classification layer. CLS-OPT performs well with far less trainable parameters.

Method	512 Dim.		1024 Dim.	
	Error	Params	Error	Params
Oracle	**95.7**	5.41M	**96.4**	10.83M
CLS-OPT	94.9	**131K**	95.8	**524K**

Table 13: Verification (%) on LFW.

Since face datasets usually contain large number of identities [17], it is natural to apply CLS-OPT to learn face embeddings. We train on CASIA [80] which has 0.5M face images of 10,572 identities, and test on LFW [25]. Since the training and testing sets do not overlap, the task well evaluates the generalizability of learned features. All methods use CNN-20 [52] and standard softmax loss. We set the output feature dimension as 512 or 1024. Table 13 validates CLS-OPT's effectiveness.

7. Concluding Remarks

We propose a novel training framework for neural networks. By parameterizing neurons with weights and a shared orthogonal matrix, OPT can provably achieve small hyperspherical energy and yield superior generalizability.

Acknowledgements. Weiyang Liu and Adrian Weller are supported by DeepMind and the Leverhulme Trust via CFI. Adrian Weller acknowledges support from the Alan Turing Institute under EPSRC grant EP/N510129/1 and U/B/000074. Rongmei Lin and Li Xiong are supported by NSF under CNS-1952192, IIS-1838200.

References

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*, 2018. 6, 36

[2] Ramesh Naidu Annavarapu. Singular value decomposition and the centrality of löwdin orthogonalizations. *American Journal of Computational and Applied Mathematics*, 3(1):33–35, 2013. 16

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *ICML*, 2016. 2, 4

[4] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? In *NeurIPS*, 2018. 2, 4, 7

[5] Johann S Brauchart, Alexander B Reznikov, Edward B Saff, Ian H Sloan, Yu Guang Wang, and Robert S Womersley. Random point sets on the sphere—hole radii, covering, and separation. *Experimental Mathematics*, 27(1):62–81, 2018. 42

[6] Anna Breger, Martin Ehler, and Manuel Gräf. Points on manifolds with asymptotically optimal covering radius. *Journal of Complexity*, 48:1–14, 2018. 42

[7] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *ICLR*, 2017. 2, 4, 7

[8] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. In *ICLR*, 2017. 5

[9] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019. 8, 20

[10] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019. 2

[11] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *ICCV*, 2019. 1, 2

[12] Simon S Du, Jason D Lee, Yuandong Tian, Barnabas Poczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. *arXiv preprint arXiv:1712.00779*, 2017. 6, 36

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 8

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 1, 3, 7

[15] Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In *NeurIPS*, 2018. 1

[16] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *NeurIPS*, 2017. 1, 6

[17] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *ECCV*, 2016. 8

[18] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 40

[19] DP Hardin and EB Saff. Minimal riesz energy point configurations for rectifiable d-dimensional manifolds. *Advances in Mathematics*, 193(1):174–204, 2005. 18

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 1, 3, 7

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6, 7, 20, 21

[22] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016. 4

[23] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 1997. 5

[24] Walter Hoffmann. Iterative algorithms for gram-schmidt orthogonalization. *Computing*, 41(4):335–348, 1989. 14

[25] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. *Technical Report*, 2008. 8

[26] Lei Huang, Li Liu, Fan Zhu, Diwen Wan, Zehuan Yuan, Bo Li, and Ling Shao. Controllable orthogonalization in training dnns. In *CVPR*, 2020. 2

[27] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *AAAI*, 2018. 2, 4

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 7

[29] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018. 5

[30] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014. 2

[31] Kui Jia, Shuai Li, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural networks. *TPAMI*, 2019. 2

[32] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *NeurIPS*, 2016. 40

[33] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *ICML*, 2017. 4

[34] Kenji Kawaguchi. Deep learning without poor local minima. In *NeurIPS*, 2016. 1, 6, 36

[35] Kenji Kawaguchi, Bo Xie, and Le Song. Deep semi-random features for nonlinear function approximation. In *AAAI*, 2018. 37

[36] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017. 1

[37] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 8

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 21

[39] N.S. Landkof. *Foundations of modern potential theory*. Springer-Verlag, 1972. 18

[40] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *COLT*, 2016. 36

[41] Mario Lezcano-Casado. Trivializations for gradient-based optimization on manifolds. In *NeurIPS*, 2019. 4

[42] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *ICML*, 2019. 2, 4

[43] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018. 5, 23, 30

[44] Jun Li, Fuxin Li, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. In *ICLR*, 2020. 4

[45] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *COLT*, 2018. 1, 6

[46] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 6, 40

[47] Rongmei Lin, Weiyang Liu, Zhen Liu, Chen Feng, Zhiding Yu, James M. Rehg, Li Xiong, and Le Song. Regularizing neural networks via minimizing hyperspherical energy. In *CVPR*, 2020. 2, 7

[48] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015. 2

[49] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. In *NeurIPS*, 2018. 2, 3, 5, 7, 8, 23, 30, 36, 43

[50] Weiyang Liu, Rongmei Lin, Zhen Liu, Li Xiong, Bernhard Schölkopf, and Adrian Weller. Learning with hyperspherical uniformity. In *AISTATS*, 2021. 2, 7

[51] Weiyang Liu, Zhen Liu, James M Rehg, and Le Song. Neural similarity learning. In *NeurIPS*, 2019. 1, 2, 40

[52] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017. 2, 8, 22

[53] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016. 8

[54] Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep hyperspherical learning. In *NeurIPS*, 2017. 2, 4, 21

[55] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, 2018. 37

[56] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *ICML*, 2017. 4

[57] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 8, 21

[58] Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. *arXiv preprint arXiv:2006.16992*, 2020. 2

[59] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, 2008. 37

[60] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019. 1

[61] Matthias Reitzner. Stochastical approximation of smooth convex bodies. *Mathematika*, 51(1-2):11–29, 2004. 42

[62] 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. *IJCV*, 2015. 7

[63] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008. 8, 21

[64] Ian H Sloan and Robert S Womersley. Extremal systems of points and numerical integration on the sphere. *Advances in Computational Mathematics*, 21(1-2):107–125, 2004. 43

[65] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017. 8

[66] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016. 36

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014. 37

[68] Vipin Srivastava. A unified view of the orthogonalization methods. *Journal of Physics A: Mathematical and General*, 33(35):6219, 2000. 16

[69] Yun Tang, Jing Huang, Guangtao Wang, Xiaodong He, and Bowen Zhou. Orthogonal relation transforms with graph context modeling for knowledge graph embedding. In *ACL*, 2020. 4

[70] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016. 8

[71] Feng Wang, Weiyang Liu, Haijun Liu, and Jian Cheng. Additive margin softmax for face verification. *arXiv preprint arXiv:1801.05599*, 2018. 2

[72] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *CVPR*, 2018. 2

[73] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *ICCV Workshops*, 2017. 2

[74] Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 2013. 4

[75] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *NeurIPS*, 2016. 2, 4

[76] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 8, 21

[77] Bo Xie, Yingyu Liang, and Le Song. Diverse neural network learns true target functions. In *AISTATS*, 2017. 6, 36, 37

[78] Di Xie, Jiang Xiong, and Shiliang Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *CVPR*, 2017. 2, 4

[79] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *ICCV*, 2015. 2

[80] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014. 8