Regularizing Neural Networks via Minimizing Hyperspherical Energy

Rongmei Lin[1], Weiyang Liu[2,*], Zhen Liu[3], Chen Feng[4], Zhiding Yu[5], James M. Rehg[2], Li Xiong[1], Le Song[2]
[1]Emory University [2]Georgia Institute of Technology [3]Mila, Université de Montréal [4]New York University [5]NVIDIA
rongmei.lin@emory.edu wyliu@gatech.edu lxiong@emory.edu lsong@cc.gatech.edu

Abstract

Inspired by the Thomson problem in physics where the distribution of multiple propelling electrons on a unit sphere can be modeled via minimizing some potential energy, hyperspherical energy minimization has demonstrated its potential in regularizing neural networks and improving their generalization power. In this paper, we first study the important role that hyperspherical energy plays in neural network training by analyzing its training dynamics. Then we show that naively minimizing hyperspherical energy suffers from some difficulties due to highly non-linear and non-convex optimization as the space dimensionality becomes higher, therefore limiting the potential to further improve the generalization. To address these problems, we propose the compressive minimum hyperspherical energy (CoMHE) as a more effective regularization for neural networks. Specifically, CoMHE utilizes projection mappings to reduce the dimensionality of neurons and minimizes their hyperspherical energy. According to different designs for the projection mapping, we propose several distinct yet well-performing variants and provide some theoretical guarantees to justify their effectiveness. Our experiments show that CoMHE consistently outperforms existing regularization methods, and can be easily applied to different neural networks.

1. Introduction

Recent years have witnessed the tremendous success of deep neural networks in a variety of tasks. With its over-parameterization nature and hierarchical structure, deep neural networks achieve unprecedented performance on many challenging problems [1, 2, 3], but their strong approximation ability also makes it easy to overfit the training set, which greatly affects the generalization on unseen samples. Therefore, how to restrict the huge parameter space and properly regularize the deep networks becomes increasingly important. Regularizations for neural networks can be roughly categorized into *implicit* and *explicit* ones. Implicit regularizations usually do not directly impose explicit constraints on neuron weights, and instead they regularize the networks in an implicit manner in order to prevent overfitting and stabilize the training. A lot of prevailing methods fall into this category, such as batch normalization [4], dropout [5], weight normalization [6], etc. Explicit regularizations [7, 8, 9, 10, 11, 12] usually introduce some penalty terms for neuron weights, and jointly optimize them along with the other objective functions.

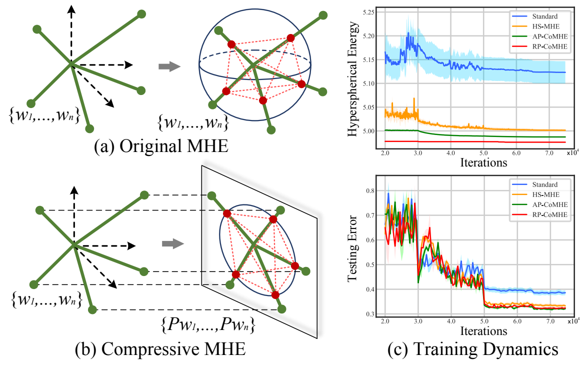

Figure 1: Comparison of original MHE and compressive MHE. In (c), the top figure shows the hyperspherical energy, and the bottom one shows the testing error (CIFAR-100). Experimental details are given in Appendix B.

Among many existing explicit regularizations, minimum hyperspherical energy (MHE) [12] stands out as a simple yet effective regularization that promotes the *hyperspherical diversity* among neurons and significantly improves the network generalization. MHE regularizes the directions of neuron weights by minimizing a potential energy on a unit hypersphere that characterizes the hyperspherical diversity (such energy is defined as *hyperspherical energy* [12]). In contrast, standard weight decay only regularizes the norm of neuron weights, which essentially can be viewed as regularizing one dimension of the weights. MHE completes an important missing piece by regularizing the neuron directions (*i.e.*, regularizing the rest dimensions of the weights).

Although minimizing hyperspherical energy has already been empirically shown useful in a number of applications [12], two fundamental questions remain unanswered: *(1) what is the role that hyperspherical energy plays in training a well-performing neural network?* and *(2) How can the hyperspherical energy be effectively minimized?* To

*Weiyang Liu is the corresponding author.

study the first question, we plot the training dynamics of hyperspherical energy (on CIFAR-100) in Fig. 1(c) for a baseline convolutional neural network (CNN) without any MHE variant, a CNN regularized by MHE [12] and a CNN regularized by our CoMHE. More experimental details and full results (with more interesting baselines) are given in Appendix B. From the empirical results in Fig. 1(c), we find that both MHE and CoMHE can achieve much lower hyperspherical energy and testing error than the baseline, showing the effectiveness of minimizing hyperspherical energy. It also implies that lower hyperspherical energy typically leads to better generalization. We empirically observe that a trained neural network with lower hyperspherical energy often generalizes better (*i.e.*, higher hyperspherical diversity leads to better generalization), and therefore we argue that hyperspherical energy is closely related to the generalization power of neural networks. In the rest of the paper, we delve into the second question that remains an open challenge: how to effectively minimize hyperspherical energy.

By adopting the definition of hyperspherical energy as the regularization objective and naively minimizing it with back-propagation, MHE suffers from a few critical problems which limit it to further unleash its potential. First, the original MHE objective has a huge number of local minima and stationary points due to its highly non-convex and non-linear objective function. The problem can get even worse when the space dimension gets higher and the number of neurons becomes larger [13, 14]. Second, the gradient of the original MHE objective *w.r.t* the neuron weight is deterministic. Unlike the weight decay whose objective is convex, MHE has a complex and non-convex regularization term. Therefore, deterministic gradients may make the solution quickly fall into one of the bad local minima and get stuck there. Third, MHE defines an ill-posed problem in general. When the number of neurons is smaller than the dimension of the space (it is often the case in neural networks), it will be less meaningful to encourage the hyperspherical diversity since the neurons can not fully occupy the space. Last, in high-dimensional spaces, randomly initialized neurons are likely to be orthogonal to each other (see Appendix C). Therefore, these high-dimensional neurons can be trivially "diverse", leading to small gradients in original MHE that cause optimization difficulties.

In order to address these problems and effectively minimize hyperspherical energy, we propose the compressive minimum hyperspherical energy (CoMHE) as a generic regularization for neural networks. The high-level intuition behind CoMHE is to project neurons to some suitable subspaces such that the hyperspherical energy can get minimized more effectively. Specifically, CoMHE first maps the neurons from a high-dimensional space to a low-dimensional one and then minimizes the hyperspherical energy of these neurons. Therefore, how to map these neurons to a low-dimensional space while preserving the desirable information in high-dimensional space is our major concern. Since we aim to regularize the directions of neurons, what we care most is the angular similarity between different neurons. To this end, we explore multiple novel methods to perform the projection and heavily study two main approaches: *random projection* and *angle-preserving projection*, which can reduce the dimensionality of neurons while still partially preserving the pairwise angles.

Random projection (RP) is a natural choice to perform the dimensionality reduction in MHE due to its simplicity and nice theoretical properties. RP can provably preserve the angular information, and most importantly, introduce certain degree of randomness to the gradients, which may help CoMHE escape from some bad local minima. The role that the randomness serves in CoMHE is actually similar to the *simulated annealing* [15, 16] that is widely used to solve Thomson problem. Such randomness is often shown to benefit the generalization [17, 18]. We also provably show that using RP can well preserve the pairwise angles between neurons. Besides RP, we propose the angle-preserving projection (AP) as an effective alternative. AP is motivated by the goal that we aim to preserve the pairwise angles between neurons. Constructing an AP that can project neurons to a low-dimensional space that well preserves the angles is often difficult even with powerful non-linear functions, which is suggested by the strong conditions required for conformal mapping in complex analysis [19]. Therefore, we frame the AP construction as an optimization problem which can be solved jointly with hyperspherical energy minimization. More interestingly, we consider the *adversarial projection* for CoMHE, which minimizes the maximal energy attained by learning the projection. We formulate it as a min-max optimization and optimize it jointly with the neural network.

However, it is inevitable to lose some information in low-dimensional spaces and the neurons may only get diverse in some particular low-dimensional spaces. To address it, we adopt multiple projections to better approximate the MHE objective in the original high-dimensional space. Specifically, we project the neurons to multiple subspaces, compute the hyperspherical energy in each space separately and then minimize the aggregation (*i.e.*, average or max). Moreover, we reinitialize these projection matrix randomly every certain number of iterations to avoid trivial solutions.

In contrast to MHE that imposes a static regularization to the neurons, CoMHE dynamically regularizes the neurons based on the projection matrices. Such dynamic regularization is equivalent to adjusting the CoMHE objective function, making it easier to escape some bad local minima. Our contributions can be summarized as:

- We first show that hyperspherical energy is closely related to generalization and then reveal the role it plays in training a neural network that generalizes well.

- To address the drawbacks of MHE, we propose CoMHE as a dynamic regularization to effectively minimize hyperspherical energy of neurons for better generalizability.
- We explore different ways to construct a suitable projection for CoMHE. Random projection and angle-preserving projection are proposed to reduce the dimensionality of neurons while preserving the angular information. We also consider several variants such as adversarial projection CoMHE and group CoMHE.
- We provide some theoretical insights for the proposed projections on the quality of preserving the angular similarity between different neurons.
- We show that CoMHE consistently outperforms the original MHE in different tasks. Notably, a 9-layer plain CNN regularized by CoMHE outperforms a standard 1001-layer ResNet by more than 2% on CIFAR-100.

2. Related Work

Diversity-based regularization has been found useful in sparse coding [20, 21], ensemble learning [22, 23], self-paced learning [24], metric learning [25], latent variable models [26], etc. Early studies in sparse coding [20, 21] model the diversity with the empirical covariance matrix and show that encouraging such diversity can improve the dictionary's generalizability. [27] promotes the uniformity among eigenvalues of the component matrix in a latent space model. [28, 29, 30, 9, 8, 30] characterize diversity among neurons with orthogonality, and regularize the neural network by promoting the orthogonality. Inspired by the Thomson problem in physics, MHE [12] defines the hyperspherical energy to characterize the diversity on a unit hypersphere and shows significant and consistent improvement in supervised learning tasks. There are two MHE variants in [12]: full-space MHE and half-space MHE. Compared to full-space MHE, the half-space variant [12] further eliminates the collinear redundancy by constructing virtual neurons with the opposite direction to the original ones and then minimizing their hyperspherical energy together. The importance of regularizing angular information is also discussed in [31, 32, 33, 34, 35, 36, 37, 38, 39, 40].

3. Compressive MHE

3.1. Revisiting Standard MHE

MHE characterizes the diversity of N neurons ($\boldsymbol{W}_N = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N \in \mathbb{R}^{d+1}\}$) on a unit hypersphere using hyperspherical energy which is defined as

$$\begin{aligned}\boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N) &= \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\left(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|\right) \\ &= \begin{cases}\sum_{i\neq j}\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-s}, & s>0\\ \sum_{i\neq j}\log\left(\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-1}\right), & s=0\end{cases}\end{aligned} \tag{1}$$

where $\|\cdot\|$ denotes ℓ_2 norm, $f_s(\cdot)$ is a decreasing real-valued function (we use $f_s(z)=z^{-s}, s>0$, *i.e.*, Riesz s-kernels), and $\hat{\boldsymbol{w}}_i = \frac{\boldsymbol{w}_i}{\|\boldsymbol{w}_i\|}$ is the i-th neuron weight projected onto the unit hypersphere $\mathbb{S}^d = \{\boldsymbol{v}\in\mathbb{R}^{d+1} | \|\boldsymbol{v}\|=1\}$. For convenience, we denote $\hat{\boldsymbol{W}}_N = \{\hat{\boldsymbol{w}}_1, \cdots, \hat{\boldsymbol{w}}_N \in \mathbb{S}^d\}$, and $\boldsymbol{E}_s = \boldsymbol{E}_{s,d}(\hat{\boldsymbol{w}}_i|_{i=1}^N)$. Note that, each neuron is a convolution kernel in CNNs. MHE minimizes the hyperspherical energy of neurons using gradient descent during back-propagation, and MHE is typically applied to the neural network in a layer-wise fashion. We first write down the gradient of $\boldsymbol{E}_2$ *w.r.t* $\hat{\boldsymbol{w}}_i$ and make the gradient to be zero:

$$\nabla_{\hat{\boldsymbol{w}}_i}\boldsymbol{E}_2 = \sum_{j=1,j\neq i}^{N}\frac{-2(\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j)}{\|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^4} = 0 \Rightarrow \hat{\boldsymbol{w}}_i = \frac{\sum_{j=1,j\neq i}^{N}\alpha_j\hat{\boldsymbol{w}}_j}{\sum_{j=1,j\neq i}^{N}\alpha_j} \tag{2}$$

where $\alpha_j = \|\hat{\boldsymbol{w}}_i - \hat{\boldsymbol{w}}_j\|^{-4}$. We use toy and informal examples to show that high dimensional space (*i.e.*, d is large) leads to much more stationary points than low-dimensional one. Assume there are $K=K_1+K_2$ stationary points in total for $\hat{\boldsymbol{W}}_N$ to satisfy Eq. 2, where K_1 denotes the number of stationary points in which every element in the solution is distinct and K_2 denotes the number of the rest stationary points. We give two examples: *(i)* For $(d+2)$-dimensional space, we can extend the solutions in $(d+1)$-dimensional space by introducing a new dimension with zero value. The new solutions satisfy Eq. 2. Because there are $d+2$ ways to insert the zero, we have at least $(d+2)K$ stationary points in $(d+2)$-dimensional space. *(ii)* We denote $K_1' = \frac{K_1}{(d+1)!}$ as the number of unordered sets that construct the stationary points. In $(2d+2)$-dimensional space, we can construct $\hat{\boldsymbol{w}}_j^E = \frac{1}{\sqrt{2}}\{\hat{\boldsymbol{w}}_j;\hat{\boldsymbol{w}}_j\}\in\mathbb{S}^{2d+1}, \forall j$ that satisfies Eq. 2. Therefore, there are at least $\frac{(2d+2)!}{2^{d+1}}K_1' + K_2$ stationary points for $\hat{\boldsymbol{W}}_N$ in $(2d+2)$-dimensional space, and besides this construction, there are much more stationary points. Therefore, MHE have far more stationary points in higher dimensions.

3.2. General Framework

To overcome MHE's drawbacks in high dimensional space, we propose the compressive MHE that projects the neurons to a low-dimensional space and then minimizes the hyperspherical energy of the projected neurons. In general, CoMHE minimizes the following form of energy:

$$\boldsymbol{E}_s^C(\hat{\boldsymbol{W}}_N) := \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\|g(\hat{\boldsymbol{w}}_i) - g(\hat{\boldsymbol{w}}_j)\|\Big) \tag{3}$$

where $g:\mathbb{S}^d\rightarrow\mathbb{S}^k$ takes a normalized $(d+1)$-dimensional input and outputs a normalized $(k+1)$-dimensional vector. $g(\cdot)$ can be either linear or nonlinear mapping. We only consider the linear case here. Using multi-layer perceptrons as $g(\cdot)$ is one of the simplest nonlinear cases. Similar to MHE, CoMHE also serves as a regularization in neural networks.

3.3. Random Projection for CoMHE

Random projection is in fact one of the most straightforward way to reduce dimensionality while partially preserving the angular information. More specifically, we use a random mapping $g(\boldsymbol{v})=\frac{\boldsymbol{P}\boldsymbol{v}}{\|\boldsymbol{P}\boldsymbol{v}\|}$ where $\boldsymbol{P}\in\mathbb{R}^{(k+1)\times(d+1)}$ is a Gaussian distributed random matrix (each entry follows i.i.d. normal distribution). In order to reduce the variance, we use C random projection matrices to project the neurons and compute the hyperspherical energy separately:

$$\boldsymbol{E}_s^R(\hat{\boldsymbol{W}}_N):=\frac{1}{C}\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|}-\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\Big\|\Big) \tag{4}$$

where $\boldsymbol{P}_c,\forall c$ is a random matrix with each entry following the normal distribution $\mathcal{N}(0,1)$. According to the properties of normal distribution [41], every normalized row of the random matrix $\boldsymbol{P}$ is uniformly distributed on a hypersphere $\mathbb{S}^d$, which indicates that the projection matrix $\boldsymbol{P}$ is able to cover all the possible subspaces. Multiple projection matrices can also be interpreted as multi-view projection, because we are making use of information from multiple projection views. In fact, we do not necessarily need to average the energy for multiple projections, and instead we can use maximum operation (or some other meaningful aggregation operations). Then the objective becomes $\max_c\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s(\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|}-\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\|)$. Considering that we aim to minimize this objective, the problem is in fact a min-max optimization. Note that, we will typically reinitialize the random projection matrices every certain number of iterations to avoid trivial solutions. Most importantly, using RP can provably preserve the angular similarity.

3.4. Angle-preserving Projection for CoMHE

Recall that we aim to find a projection to project the neurons to a low-dimensional space that best preserves angular information. We transform the goal to an optimization:

$$\boldsymbol{P}^\star=\arg\min_{\boldsymbol{P}}\mathcal{L}_P:=\sum_{i\neq j}(\theta_{(\hat{\boldsymbol{w}}_i,\hat{\boldsymbol{w}}_j)}-\theta_{(\boldsymbol{P}\hat{\boldsymbol{w}}_i,\boldsymbol{P}\hat{\boldsymbol{w}}_j)})^2 \tag{5}$$

where $\boldsymbol{P}\in\mathbb{R}^{(k+1)\times(d+1)}$ is the projection matrix and $\theta_{(\boldsymbol{v}_1,\boldsymbol{v}_2)}$ denotes the angle between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. For implementation convenience, we can replace the angle with the cosine value (*e.g.*, use $\cos(\theta_{(\hat{\boldsymbol{w}}_i,\hat{\boldsymbol{w}}_j)})$ to replace $\theta_{(\hat{\boldsymbol{w}}_i,\hat{\boldsymbol{w}}_j)}$), so that we can directly use the inner product of normalized vectors to measure the angular similarity. With $\hat{\boldsymbol{P}}$ obtained in Eq. 5, we use a nested loss function:

$$\begin{aligned}\boldsymbol{E}_s^A(\hat{\boldsymbol{W}}_N,\boldsymbol{P}^\star)&:=\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}^\star\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}^\star\hat{\boldsymbol{w}}_i\|}-\frac{\boldsymbol{P}^\star\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}^\star\hat{\boldsymbol{w}}_j\|}\Big\|\Big)\\ \text{s.t.}\quad \boldsymbol{P}^\star&=\arg\min_{\boldsymbol{P}}\sum_{i\neq j}(\theta_{(\hat{\boldsymbol{w}}_i,\hat{\boldsymbol{w}}_j)}-\theta_{(\boldsymbol{P}\hat{\boldsymbol{w}}_i,\boldsymbol{P}\hat{\boldsymbol{w}}_j)})^2\end{aligned} \tag{6}$$

for which we propose two different ways to optimize the projection matrix $\boldsymbol{P}$. We can approximate $\boldsymbol{P}^\star$ using a few gradient descent updates. Specifically, we use two different ways to perform the optimization. Naively, we use a few gradient descent steps to update $\boldsymbol{P}$ in order to approximate $\boldsymbol{P}^\star$ and then update $\boldsymbol{W}_N$, which proceeds alternately. The number of iteration steps that we use to update $\boldsymbol{P}$ is a hyperparemter and needs to be determined by cross-validation. Besides the naive alternate one, we also use a different optimization of $\boldsymbol{W}_N$ by unrolling the gradient update of $\boldsymbol{P}$.

Alternating optimization. The alternating optimization is to optimize $\boldsymbol{P}$ alternately with the network parameters $\boldsymbol{W}_N$. Specifically, in each iteration of updating the network parameters, we update $\boldsymbol{P}$ every number of inner iterations and use it as an approximation to $\boldsymbol{P}^\star$ (the error depends on the number of gradient steps we take). Essentially, we are alternately solving two separate optimization problems for $\boldsymbol{P}$ and $\boldsymbol{W}_N$ with gradient descent.

Unrolled optimization. Instead of naively updating $\boldsymbol{W}_N$ with approximate $\boldsymbol{P}^\star$ in the alternating optimization, the unrolled optimization further unrolls the update rule of $\boldsymbol{P}$ and embed it within the optimization of network parameters $\boldsymbol{W}_N$. If we denote the CoMHE loss with a given projection matrix $\boldsymbol{P}$ as $\boldsymbol{E}_s^A(\boldsymbol{W}_N,\boldsymbol{P})$ which takes $\boldsymbol{W}_N$ and $\boldsymbol{P}$ as input, then the unrolled optimization is essentially optimizing $\boldsymbol{E}_s^A(\boldsymbol{W}_N,\boldsymbol{P}-\eta\cdot\frac{\partial\mathcal{L}_P}{\partial\boldsymbol{P}})$. It can also be viewed as minimizing the CoMHE loss after a single step of gradient descent *w.r.t.* the projection matrix. This optimization includes the computation of second-order partial derivatives. Note that, it is also possible to unroll multiple gradient descent steps. Similar unrolling is also applied in [42, 43, 44].

3.5. Notable CoMHE Variants

We provide more interesting CoMHE variants as an extension. We will have some preliminary empirical study on these variants, but our main focus is still on RP and AP.

Adversarial Projection for CoMHE. We consider a novel CoMHE variant that adversarially learns the projection. The intuition behind is that we want to learn a projection basis that maximizes the hyperspherical energy while the final goal is to minimize this maximal energy. With such intuition, we can construct a min-max optimization:

$$\min_{\hat{\boldsymbol{W}}_N}\max_{\boldsymbol{P}}\boldsymbol{E}_s^V(\hat{\boldsymbol{W}}_N,\boldsymbol{P}):=\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}\hat{\boldsymbol{w}}_i\|}-\frac{\boldsymbol{P}\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}\hat{\boldsymbol{w}}_j\|}\Big\|\Big) \tag{7}$$

which can be solved by gradient descent similar to [45]. From a game-theoretical perspective, $\boldsymbol{P}$ and $\hat{\boldsymbol{W}}_N$ can be viewed as two players that are competing with each other. However, due to the instability of solving the min-max problem, the performance of this projection is unstable.

Group CoMHE. Group CoMHE is a very special case in the CoMHE framework. The basic idea is to divide the weights of each neuron into several groups and then minimize the hyperspherical energy within each group. For example in CNNs, group MHE divides the channels into groups and minimizes within each group the MHE loss. Specifically, the objective function of group CoMHE is

$$\boldsymbol{E}_s^G(\hat{\boldsymbol{W}}_N):=\frac{1}{C}\sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} f_s\Big(\Big\|\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_i}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_i\|}-\frac{\boldsymbol{P}_c\hat{\boldsymbol{w}}_j}{\|\boldsymbol{P}_c\hat{\boldsymbol{w}}_j\|}\Big\|\Big) \tag{8}$$

where $\boldsymbol{P}_c$ is a diagonal matrix with every diagonal entry being either 0 or 1, and $\sum_c\boldsymbol{P}_c=\boldsymbol{I}$ (in fact, this is optional).

There are multiple ways to divide groups for the neurons, and typically we will divide groups according to the channels, similar to [46]. More interestingly, one can also divide the groups in a *stochastic* fashion.

3.6. Shared Projection Basis in Neural Networks

In general, we usually need different projection bases for neurons in different layers of the neural network. However, we find it beneficial to share some projection bases across different layers. We only share the projection matrix for the neurons in different layers that have the same dimensionality. For example in a neural network, if the neurons in the first layer have the same dimensionality with the neurons in the second layer, we will share their projection matrix that reduces the dimensionality. Sharing the projection basis can effectively reduce the number of projection parameters and may also reduce the inconsistency within the hyperspherical energy minimization of projected neurons in different layers. Most importantly, it can empirically improve the network generalizability while using much fewer parameters and saving more computational overheads.

4. Theoretical Insights

4.1. Angle Preservation

We start with highly relevant properties of random projection and then delve into the angular preservation.

Lemma 1 (Mean Preservation of Random Projection)**.** *For any $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$ and any random Gaussian distributed matrix $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}} r_{ij}$, if $r_{ij}, \forall i, j$ are i.i.d. random variables from $\mathcal{N}(0,1)$, we have $\mathbb{E}(\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2\rangle) = \langle \boldsymbol{w}_1, \boldsymbol{w}_2\rangle$.*

This lemma indicates that the mean of randomly projected inner product is well preserved, partially justifying why using random projection actually makes senses.

Johnson-Lindenstrauss lemma (JLL) [47, 48] (in Appendix D) establishes a guarantee for the Euclidean distance between randomly projected vectors. However, JLL does not provide the angle preservation guarantees. It is nontrivial to provide a guarantee for angular similarity from JLL.

Theorem 1 (Angle Preservation I)**.** *Given $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ is a random projection matrix that has i.i.d. 0-mean σ-subgaussian entries, and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ are the randomly projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under $\boldsymbol{P}$. Then $\forall \epsilon \in (0,1)$, we have that*

$$\frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) - \epsilon}{1+\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) < \frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) + \epsilon}{1-\epsilon} \tag{9}$$

which holds with probability $\left(1 - 2\exp(-\frac{k\epsilon^2}{8})\right)^2$.

Theorem 2 (Angle Preservation II)**.** *Given $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ is a Gaussian random projection matrix where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}} r_{ij}$ ($r_{ij}, \forall i, j$ are i.i.d. random variables from $\mathcal{N}(0,1)$), and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ are the randomly projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under $\boldsymbol{P}$. Then $\forall \epsilon \in (0,1)$ and $\boldsymbol{w}_1^\top \boldsymbol{w}_2 > 0$, we have that*

$$\begin{aligned} \frac{1+\epsilon}{1-\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) - \frac{2\epsilon}{1-\epsilon} &< \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) \\ &< \frac{1-\epsilon}{1+\epsilon}\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}) + \frac{1+2\epsilon}{1+\epsilon} - \frac{\sqrt{(1-\epsilon^2)}}{1+\epsilon} \end{aligned} \tag{10}$$

which holds with probability $1 - 6\exp(-\frac{k}{2}(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}))$.

Theorem 1 is one of our main theoretical results and reveals that the angle between randomly projected vectors is well preserved. Note that, the parameter σ of the subgaussian distribution is not related to our bound for the angle, so any Gaussian distributed random matrix has the property of angle preservation. The projection dimension k is related to the probability that the angle preservation bound holds. Theorem 2 is a direct result from [49]. It again shows that the angle between randomly projected vectors is provably preserved. Both Theorem 1 and Theorem 2 give upper and lower bounds for the angle between randomly projected vectors. If $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)} > \arccos(\frac{\epsilon+3\epsilon^2}{3\epsilon+\epsilon^2})$, then the lower bound in Theorem 1 is tighter than the lower bound in Theorem 2. If $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)} > \arccos(\frac{1-3\epsilon^2-(1-\epsilon)\sqrt{1-\epsilon^2}}{3\epsilon-\epsilon^2})$, the upper bound in Theorem 1 is tighter than the upper bound in Theorem 2. To conclude, Theorem 1 gives tighter bounds when the angle of original vectors is large. Since AP is randomly initialized every certain number of iterations and minimizes the angular difference before and after the projection, AP usually performs better than RP in preserving angles. Without the angle-preserving optimization, AP reduces to RP.

4.2. Statistical Insights

We can also draw some theoretical intuitions from spherical uniform testing [50] in statistics. Spherical uniform testing is a nonparametric statistical hypothesis test that checks whether a set of observed data is generated from a uniform distribution on a hypersphere or not. Random projection is in fact an important tool [50] in statistics to test the uniformity on hyperspheres, while our goal is to promote the same type of hyperspherical uniformity (*i.e.*, diversity). Specifically, we have N random samples $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N$ of $\mathbb{S}^d$-valued random variables, and the random projection $\boldsymbol{p}$ which is another random variable independent of $\boldsymbol{w}_i, \forall i$ and uniformly distributed on $\mathbb{S}^d$. The projected points of $\boldsymbol{w}_i, \forall i$ is $y_i = \boldsymbol{p}^\top \boldsymbol{w}_i, \forall i$. The distribution of $y_i, \forall i$ uniquely determines the distribution of $\boldsymbol{w}_1$, as is specified by Theorem 3.

Theorem 3 (Unique Distribution Determination of Random Projection)**.** *Let $\boldsymbol{w}$ be a $\mathbb{S}^d$-valued random variable and $\boldsymbol{p}$ be a random variable that is uniformly distributed on $\mathbb{S}^d$ and independent of $\boldsymbol{w}$. With probability one, the distribution of $\boldsymbol{w}$ is uniquely determined by the distribution of the projection of $\boldsymbol{w}$ on $\boldsymbol{p}$. More specifically, if $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are $\mathbb{S}^d$-valued random variables, independent of $\boldsymbol{p}$ and we have* *a positive probability for the event that $\boldsymbol{p}$ takes a value $\boldsymbol{p}_0$ such that the two distributions satisfy $\boldsymbol{p}_0^\top \boldsymbol{w}_1 \sim \boldsymbol{p}_0^\top \boldsymbol{w}_2$, then $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are identically distributed.*

Theorem 3 shows that the distributional information is well preserved after random projection, providing the CoMHE framework a statistical intuition and foundation. We emphasize that the randomness here is in fact very crucial. For a fixed projection $\boldsymbol{p}_0$, Theorem 3 does not hold in general. As a result, random projection for CoMHE is well motivated from the statistical perspective.

4.3. Insights from Random Matrix Theory

Random projection may also impose some implicit regularization to learning the neuron weights. [51] proves that random projection serves as a regularizer for the Fisher linear discrimination classifier. From metric learning perspective, the inner product between neurons $\boldsymbol{w}_1^\top \boldsymbol{w}_2$ will become $\boldsymbol{w}_1^\top \boldsymbol{P}^\top \boldsymbol{P} \boldsymbol{w}_2$ where $\boldsymbol{P}^\top \boldsymbol{P}$ defines a specific form of (low-rank) similarity [52, 39]. [53] proves that random projection satisfying the JLL *w.h.p* also satisfies the restricted isometry property (RIP) *w.h.p* under sparsity assumptions. In this case, the neuron weights can be well recovered [54, 55]. These results imply that randomly projected neurons in CoMHE may implicitly regularize the network.

5. Discussions and Extensions

Bilateral projection for CoMHE. If we view the neurons in one layer as a matrix $\boldsymbol{W} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n\} \in \mathbb{R}^{m\times n}$ where m is the dimension of neurons and n is the number of neurons, then the projection considered throughout the paper is to left-multiply a projection matrix $\boldsymbol{P}_1 \in \mathbb{R}^{r\times m}$ to $\boldsymbol{W}$. In fact, we can further reduce the number of neurons by right-multiplying an additional projection matrix $\boldsymbol{P}_2 \in \mathbb{R}^{n\times r}$ to $\boldsymbol{W}$. Specifically, we denote that $\boldsymbol{Y}_1 = \boldsymbol{P}_1 \boldsymbol{W}$ and $\boldsymbol{Y}_2 = \boldsymbol{W}\boldsymbol{P}_2$. Then we can apply the MHE regularization separately to column vectors of $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$. The final neurons are still $\boldsymbol{W}$. More interestingly, we can also approximate $\boldsymbol{W}$ with a low-rank factorization [56]: $\tilde{\boldsymbol{W}} = \boldsymbol{Y}_2(\boldsymbol{P}_1\boldsymbol{Y}_2)^{-1}\boldsymbol{Y}_1 \in \mathbb{R}^{m\times n}$. It inspires us to directly use two set of parameters $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to represent the equivalent neurons $\tilde{\boldsymbol{W}}$ and apply the MHE regularization separately to their column vectors. Different from the former case, we use $\tilde{\boldsymbol{W}}$ as the final neurons. More details are in Appendix F.

Constructing random projection matrices. In random projection, we typically construct random matrices with each element drawn *i.i.d.* from a normal distribution. However, there are many more choices for constructing a random matrices that can provably preserve distance information. For example, we have subsampled randomized Hadamard transform [57] and count sketch-based projections [58].

Comparison to existing works. One of the widely used regularizations is the orthonormal regularization [32, 59] that minimizes $\|\boldsymbol{W}^\top\boldsymbol{W} - \boldsymbol{I}\|_F$ where W denotes the weights of a group of neurons with each column being one neuron and $\boldsymbol{I}$ is an identity matrix. [9, 29] are also built upon orthogonality. In contrast, both MHE and CoMHE do not encourage orthogonality among neurons and instead promote hyperspherical uniformity and diversity.

Randomness improves generalization. Both RP and AP introduce randomness to CoMHE, and the empirical results show that such randomness can greatly benefit the network generalization. It is well-known that stochastic gradient is one of the key ingredients that help neural networks generalize well to unseen samples. Interestingly, randomness in CoMHE also leads to a stochastic gradient. [17] also theoretically shows that randomness helps generalization, partially justifying the effectiveness of CoMHE.

6. Experiments and Results

6.1. Image Recognition

We perform image recognition to show the improvement of regularizing CNNs with CoMHE. Our goal is to show the superiority of CoMHE rather than achieving state-of-the-art accuracies on particular tasks. For all the experiments on CIFAR-10 and CIFAR-100 in the paper, we use the same data augmentation as [1, 34]. For ImageNet-2012, we use the same data augmentation in [32]. We train all the networks using SGD with momentum 0.9. All the networks use BN [4] and ReLU if not otherwise specified. By default, all CoMHE variants are built upon half-space MHE. Experimental details are given in each subsection and Appendix A. More experiments are given in Appendix I,H,J.

6.1.1 Ablation Study and Exploratory Experiments

Method	Error (%)
Baseline	28.03
Orthogonal	27.01
SRIP [9]	25.80
MHE [12]	26.75
HS-MHE [12]	25.96
G-CoMHE	25.08
Adv-CoMHE	25.09
RP-CoMHE	24.39
RP-CoMHE (max)	24.77
AP-CoMHE (alter.)	24.95
AP-CoMHE (unroll)	**24.33**

Table 1: CoMHE variants on C-100.

Variants of CoMHE. We compare different variants of CoMHE with the same plain CNN-9 (Appendix A). Specifically, we evaluate the baseline CNN without any regularization, half-space MHE (HS-MHE) which is the best MHE variant from [12], random projection CoMHE (RP-CoMHE), RP-CoMHE (max) that uses max instead of average for loss aggregation, angle-preserving projection CoMHE (AP-CoMHE), adversarial projection CoMHE (Adv-CoMHE) and group CoMHE (G-CoMHE) on CIFAR-100. For RP, we set the projection dimension to 30 (*i.e.*, $k = 29$) and the number of projection to 5 (*i.e.*, $C = 5$). For AP, the number of projection is 1 and the projection dimension is set to 30. For AP, we evaluate both alternating optimization and unrolled optimization. In alternating optimization, we update the projection matrix every 10 steps of network update. In unrolled optimization, we only unroll one-step gradient in the optimization. For G-CoMHE, we construct a group with every 8 consecutive channels. All these design choices are obtained using cross-validation. We will also study how these hyperparameters affect the performance in the following experiments. The results in Table 1 show that all of our proposed CoMHE variants can outperform the original half-space MHE by a large margin. The unrolled optimization in AP-CoMHE shows the significant advantage over alternating one and achieves the best accuracy. Both Adv-CoMHE and G-CoMHE achieve decent performance gain over HS-MHE, but not as good as RP-CoMHE and AP-CoMHE. Therefore, we will mostly focus on RP-CoMHE and AP-CoMHE in the remaining experiments.

Projection Dimension	10	20	30	40	80
RP-CoMHE	25.48	25.32	24.60	**24.75**	25.46
AP-CoMHE (alter.)	**25.21**	24.60	24.95	24.97	**24.99**
AP-CoMHE (unroll.)	25.32	**24.59**	**24.33**	24.93	25.12

Table 2: Error (%) on CIFAR-100 under different dimension of projection.

Dimension of projection. We evaluate how the dimension of projection (*i.e.*, k) affects the performance. We use the plain CNN-9 as the backbone network and test on CIFAR-100. We fix the number of projections in RP-CoMHE to 20. Because AP-CoMHE does not need to use multiple projections to reduce variance, we only use one projection in AP-CoMHE. Results are given in Table 2. In general, RP-CoMHE and AP-CoMHE with different projection dimensions can consistently and significantly outperform the half-space MHE, validating the effectiveness and superiority of the proposed CoMHE framework. Specifically, we find that both RP-CoMHE and AP-CoMHE usually achieve the best accuracy when the projection dimension is 20 or 30. Since the unrolled optimization in AP-CoMHE is consistently better than the alternating optimization, we stick to the unrolled optimization for AP-CoMHE in the remaining experiments if not otherwise specified.

Number of projections. We evaluate RP-CoMHE under different numbers of projections. We use the plain CNN-9 as the baseline and test on CIFAR-100. Results in Table 3 show that the performance of RP-CoMHE is generally not very sensitive to the number of projections. Surprisingly, we find that it is not necessarily better to use more projections for variance reduction. Our experiment show that using 5 projections can achieve the best accuracy. It may be because large variance can help the solution escape bad local minima in the optimization. Note that, we generally do not use multiple projections in AP-CoMHE, because AP-CoMHE optimizes the projection and variance reduction is unnecessary. Our results do not show performance gain by using multiple projections in AP-CoMHE.

# Proj.	RP-CoMHE	AP-CoMHE
1	25.11	**24.33**
5	**24.39**	24.34
10	25.11	24.36
20	24.60	24.38
30	24.82	24.52
80	24.92	24.56

Table 3: Error (%) on CIFAR-100 under different numbers of projections.

Width	$t=1$	$t=2$	$t=4$	$t=8$	$t=16$	$t=20$
Baseline	47.72	38.64	28.13	24.95	24.44	23.77
MHE [12]	36.84	30.05	26.75	24.05	23.14	22.36
HS-MHE [12]	35.16	29.33	25.96	23.38	21.83	21.22
RP-CoMHE	**34.73**	**28.92**	24.39	**22.44**	20.81	20.62
AP-CoMHE	34.89	29.01	**24.33**	22.6	**20.72**	**20.50**

Table 4: Error (%) on CIFAR-100 with different network width.

Network width. We evaluate RP-CoMHE and AP-CoMHE with different network width on CIFAR-100. We use the plain CNN-9 as our backbone network architecture, and set its filter number in Conv1.x, Conv2.x and Conv3.x (see Appendix A) to $16 \times t$, $32 \times t$ and $64 \times t$, respectively. Specifically, we test the cases where $t = 1, 2, 4, 8, 16$. Taking $t = 2$ as an example, then the filter numbers in Conv1.x, Conv2.x and Conv3.x are 32, 64 and 128, respectively. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. The results are shown in Table 4. Note that, we use the unrolled optimization in AP-CoMHE. From Table 4, one can observe that the performance gains of both RP-CoMHE and AP-CoMHE are very consistent and significant. With wider network, CoMHE also achieves better accuracy. Compared to the strong results of half-space MHE, CoMHE still obtains more than 1% accuracy boost under different network width.

Network depth. We evaluate RP-CoMHE and AP-CoMHE with different network depth on CIFAR-100. We use three plain CNNs with 6, 9 and 15 convolution layers, respectively. For all the networks, we set the filter number in Conv1.x, Conv2.x and Conv3.x to 64, 128 and 256, respectively. Detailed network architectures are given in Appendix A. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. Table 5 shows that both RP-CoMHE and AP-CoMHE can outperform half-space MHE by a considerable margin while regularizing a plain CNN with different depth.

Depth	CNN-6	CNN-9	CNN-15
Baseline	32.08	28.13	N/C
MHE [12]	28.16	26.75	26.90
HS-MHE [12]	27.56	25.96	25.84
RP-CoMHE	26.73	24.39	**24.21**
AP-CoMHE	**26.55**	**24.33**	24.55

Table 5: Error on CIFAR-100 with different network depth. N/C denotes Not Converged.

Effectiveness of optimization. To verify that our CoMHE can better minimize the hyperspherical energy, we compute the hyperspherical energy $\boldsymbol{E}_2$ (Eq. 1) for baseline CNN and CNN regularized by orthogonal regularization, HS-MHE, RP-CoMHE and AP-CoMHE during training. Note that, we compute the original hyperspherical energy rather than the energy after projection. All

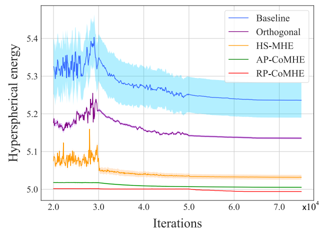

Figure 2: Hyperspherical energy during training. All networks are initialized with the same random weights, so the hyperspherical energy is the same before the training starts.

the networks use exactly the same initialization (the initial hyperspherical energy is the same). The results are averaged over five independent runs. We show the hyperspherical energy after the 20000-th iteration, because at the beginning of the training, hyperspherical energy fluctuates dramatically and is unstable. Interested readers can refer to Appendix G for the complete energy dynamics. From Fig. 2, one can observe that both RP-CoMHE and AP-CoMHE can better minimize the hyperspherical energy. RP-CoMHE can achieve the lowest energy with smallest standard deviation. From the absolute scale, the optimization gain is also very significant. In the high-dimensional space, the variance of hyperspherical energy is usually small (already close to the smallest energy value) and is already difficult to minimize.

ResNet with CoMHE. All the above experiments are performed using VGG-like plain CNNs, so we use the more powerful ResNet [1] to show that CoMHE is architecture-agnostic. We use the same experimental setting in [60] for fair comparison. We use a standard ResNet-32 as our baseline and the network architecture is specified in Appendix A. From the results in Table 6, one can observe that both RP-CoMHE and AP-CoMHE can consistently outperform half-space MHE, showing that CoMHE can boost the performance across different network architectures. More interestingly, the ResNet-32 regularized by CoMHE achieves impressive accuracy and is able to outperform the 1001-layer ResNet by a large margin. Additionally, we note that from Table 4, we can regularize a plain VGG-like 9-layer CNN with CoMHE and achieve 20.81% error rate, which is nearly 2% improvement over the 1001-layer ResNet.

Method	C-10	C-100
ResNet-110 [1]	6.61	25.16
ResNet-1001 [60]	4.92	**22.71**
Baseline	5.19	22.87
Orthogonal [29]	5.02	22.36
SRIP [9]	4.75	22.08
MHE [12]	4.72	22.19
HS-MHE [12]	4.66	22.04
RP-CoMHE	4.59	21.82
AP-CoMHE	**4.57**	**21.63**

Table 6: Error (%) using ResNets.

6.1.2 Large-scale Recognition on ImageNet-2012

We evaluate CoMHE for image recognition on ImageNet-2012 [61]. We perform the experiment using ResNet-18, ResNet-34 and ResNet-50, and then report the top-1 validation error (center crop) in Table 7. Our results show consistent and significant performance gain of CoMHE in all ResNet variants. Compared to the baselines, CoMHE can reduce the top-1 error for more than 1%. Since the computational overhead of CoMHE is almost neglectable, the performance gain is obtained without many efforts. Most importantly, as a plug-in regularization, CoMHE is shown to be architecture-agnostic and produces considerable accuracy gain in most circumstances.

Method	Res-18	Res-34	Res-50
baseline	32.95	30.04	25.30
Orthogonal [29]	32.65	29.74	25.19
Orthnormal [32]	32.61	29.75	25.21
SRIP [9]	32.53	29.55	24.91
MHE [12]	32.50	29.60	25.02
HS-MHE [12]	32.45	29.50	24.98
RP-CoMHE	31.90	29.38	**24.51**
AP-CoMHE	**31.80**	**29.32**	24.53

Table 7: Top-1 center crop error on ImageNet.

Besides the accuracy improvement, we also visualize in Fig. 3 the 64 filters in the first-layer learned by the baseline ResNet and the CoMHE-regularized ResNet. The filters look quite different after we regularize the network using CoMHE. Each filter learned by baseline focuses on a particular local pattern (*e.g.*, edge, color and shape) and each one has a clear local semantic meaning. In contrast, filters learned by CoMHE focuses more on edges, textures and global patterns which do not necessarily have a clear local semantic meaning. However, from a representation basis perspective, having such global patterns may be beneficial to the recognition accuracy. We also observe that filters learned by CoMHE pay less attention to color.

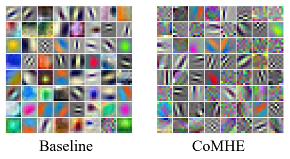

Figure 3: Visualized first-layer filters.

6.2. Point Cloud Recognition

We evaluate CoMHE on point cloud recognition. Our goal is to validate the effectiveness of CoMHE on a totally different network architecture with a different form of input data structure, rather than achieving state-of-the-art performance on point cloud recognition. To this end, we conduct experiments on widely used neural networks that handles point clouds: PointNet [62] (PN) and PointNet++ [63] (PN++). We combine half-space MHE, RP-CoMHE and AP-CoMHE into PN (without T-Net), PN (with T-Net) and PN++. More experimental details are given in Appendix A. We test the performance on ModelNet-40 [64]. Specifically, since PN can be viewed as 1×1 convolutions before the max pooling layer, we can apply all these MHE variants similarly to CNN. After the max pooling layer, there is a standard fully connected network where we can still apply the MHE variants. We compare the performance of regularizing PN and PN++ with half-space MHE, RP-CoMHE or AP-CoMHE. Table 8 shows that all MHE variants consistently improve PN and PN++, while RP-CoMHE and AP-CoMHE again perform the best among all. We demonstrate that CoMHE is generally useful for different types of input data (not limit to images) and different types of neural networks. CoMHE is also useful in graph neural networks (Appendix J).

Method	PN	PN (T)	PN++
Original	87.1	89.20	90.07
MHE [12]	87.31	89.33	90.25
HS-MHE [12]	87.44	89.41	90.31
RP-CoMHE	87.82	89.69	90.52
AP-CoMHE	**87.85**	**89.70**	**90.56**

Table 8: Accuracy (%) on ModelNet-40.

7. Concluding Remarks

Since naively minimizing hyperspherical energy yields suboptimal solutions, we propose a novel framework which projects the neurons to suitable spaces and minimizes the energy there. Experiments validate CoMHE's superiority.

Acknowledgements. The research is partially supported by NSF BigData program under IIS-1838200, NSF CPS program under CMMI-1932187, and USDOT via C2SMART under 69A3551747124.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6, 8, 12

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 6

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014. 1

[6] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 1

[7] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 1

[8] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *ICLR*, 2016. 1, 3

[9] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? In *NeurIPS*, 2018. 1, 3, 6, 8

[10] Pau Rodríguez, Jordi Gonzalez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*, 2016. 1

[11] 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. 1

[12] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. *NeurIPS*, 2018. 1, 2, 3, 6, 7, 8, 12, 13, 15, 25

[13] J Batle, Armen Bagdasaryan, M Abdel-Aty, and S Abdalla. Generalized thomson problem in arbitrary dimensions and non-euclidean geometries. *Physica A: Statistical Mechanics and its Applications*, 451:237–250, 2016. 2

[14] Matthew Calef, Whitney Griffiths, and Alexia Schulz. Estimating the number of stable configurations for the generalized thomson problem. *Journal of Statistical Physics*, 160(1):239–253, 2015. 2

[15] Y Xiang, DY Sun, W Fan, and XG Gong. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220, 1997. 2

[16] Y Xiang and XG Gong. Efficiency of generalized simulated annealing. *Physical Review E*, 62(3):4473, 2000. 2

[17] Kenji Kawaguchi, Bo Xie, and Le Song. Deep semi-random features for nonlinear function approximation. In *AAAI*, 2018. 2, 6

[18] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2008. 2

[19] Zeev Nehari. *Conformal mapping*. Courier Corporation, 2012. 2

[20] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *ICML*, 2009. 3

[21] Ignacio Ramirez, Pablo Sprechmann, and Guillermo Sapiro. Classification and clustering via dictionary learning with structured incoherence and shared features. In *CVPR*, 2010. 3

[22] Nan Li, Yang Yu, and Zhi-Hua Zhou. Diversity regularized ensemble pruning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2012. 3

[23] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003. 3

[24] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *NIPS*, 2014. 3

[25] Pengtao Xie, Wei Wu, Yichen Zhu, and Eric P Xing. Orthogonality-promoting distance metric learning: convex relaxation and theoretical analysis. In *ICML*, 2018. 3

[26] Pengtao Xie, Jun Zhu, and Eric Xing. Diversity-promoting bayesian learning of latent variable models. In *ICML*, 2016. 3

[27] Pengtao Xie, Aarti Singh, and Eric P Xing. Uncorrelation and evenness: a new diversity-promoting regularizer. In *ICML*, 2017. 3

[28] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. In *ICLR*, 2016. 3

[29] Pau Rodríguez, Jordi Gonzalez, Guillem Cucurull, Josep M Gonfaus, and Xavier Roca. Regularizing cnns with locally constrained decorrelations. In *ICLR*, 2017. 3, 6, 8

[30] 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. *arXiv:1703.01827*, 2017. 3

[31] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016. 3

[32] Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep hyperspherical learning. In *NIPS*, 2017. 3, 6, 8, 12

[33] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017. 3

[34] Weiyang Liu, Zhen Liu, Zhiding Yu, Bo Dai, Rongmei Lin, Yisen Wang, James M Rehg, and Le Song. Decoupled networks. *CVPR*, 2018. 3, 6

[35] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *arXiv preprint arXiv:1801.07698*, 2018. 3

[36] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. *arXiv preprint arXiv:1801.09414*, 2018. 3

[37] Feng Wang, Xiang Xiang, Jian Cheng, and Alan L Yuille. Normface: L2 hypersphere embedding for face verification. *arXiv preprint arXiv:1704.06369*, 2017. 3

[38] Feng Wang, Weiyang Liu, Haijun Liu, and Jian Cheng. Additive margin softmax for face verification. *arXiv preprint arXiv:1801.05599*, 2018. 3

[39] Weiyang Liu, Zhen Liu, James M Rehg, and Le Song. Neural similarity learning. In *NeurIPS*, 2019. 3, 6

[40] Pascal Mettes, Elise van der Pol, and Cees Snoek. Hyperspherical prototype networks. In *NeurIPS*, 2019. 3

[41] Harald Cramér. *Mathematical methods of statistics (PMS-9)*, volume 9. Princeton university press, 2016. 4

[42] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 4

[43] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 4

[44] Bo Dai, Hanjun Dai, Niao He, Weiyang Liu, Zhen Liu, Jianshu Chen, Lin Xiao, and Le Song. Coupled variational bayes via optimization embedding. In *NIPS*, 2018. 4

[45] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 4

[46] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 5

[47] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003. 5, 17

[48] Ata Kaban. Improved bounds on the dot product under random projection and random sign projection. In *KDD*, 2015. 5, 17

[49] Qinfeng Shi, Chunhua Shen, Rhys Hill, and Anton van den Hengel. Is margin preserved after random projection? *arXiv preprint arXiv:1206.4651*, 2012. 5, 18

[50] Juan A Cuesta-Albertos, Antonio Cuevas, and Ricardo Fraiman. On projection-based tests for directional and compositional data. *Statistics and Computing*, 19(4):367, 2009. 5

[51] Robert J Durrant and Ata Kabán. Random projections as regularizers: learning a linear discriminant from fewer observations than dimensions. *Machine Learning*, 2015. 6

[52] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2003. 6

[53] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 2008. 6

[54] Yaniv Plan and Roman Vershynin. One-bit compressed sensing by linear programming. *Communications on Pure and Applied Mathematics*, 2013. 6

[55] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 2006. 6

[56] Tianyi Zhou and Dacheng Tao. Godec: Randomized low-rank & sparse matrix decomposition in noisy case. In *ICML*, 2011. 6, 20

[57] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *SOTC*, 2006. 6

[58] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 2004. 6

[59] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016. 6

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 8

[61] 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*, pages 1–42, 2014. 8

[62] 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, 13

[63] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 8, 13

[64] 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

[65] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 12

[66] Weiyang Liu, Rongmei Lin, Zhen Liu, Li Xiong, and Le Song. Orthogonal over-parameterized training. *Technical Report*, 2020. 14, 21

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 14

[68] Juan Antonio Cuesta-Albertos, Ricardo Fraiman, and Thomas Ransford. A sharp form of the cramer–wold theorem. *Journal of Theoretical Probability*, 20(2):201–209, 2007. 19

[69] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 25

Appendix

A. Experimental Details

Layer	CNN-6	CNN-9	CNN-15
Conv1.x	[3×3, 64]×2	[3×3, 64]×3	[3×3, 64]×5
Pool1	2×2 Max Pooling, Stride 2		
Conv2.x	[3×3, 128]×2	[3×3, 128]×3	[3×3, 128]×5
Pool2	2×2 Max Pooling, Stride 2		
Conv3.x	[3×3, 256]×2	[3×3, 256]×3	[3×3, 256]×5
Pool3	2×2 Max Pooling, Stride 2		
Fully Connected	256	256	256

Table 9: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

Layer	ResNet-32 for CIFAR-10/100	ResNet-18 for ImageNet-2012	ResNet-34 for ImageNet-2012
Conv0.x	N/A	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2	[7×7, 64], Stride 2 3×3, Max Pooling, Stride 2
Conv1.x	[3×3, 64]×1 $\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times5$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$
Conv2.x	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times5$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$
Conv3.x	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times5$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$
Conv4.x	N/A	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$
	Average Pooling		

Table 10: Our ResNet architectures with different convolutional layers. Conv0.x, Conv1.x, Conv2.x, Conv3.x and Conv4.x denote convolution units that may contain multiple convolutional layers, and residual units are shown in double-column brackets. Conv1.x, Conv2.x and Conv3.x usually operate on different size feature maps. These networks are essentially the same as [1], but some may have a different number of filters in each layer. The downsampling is performed by convolutions with a stride of 2. E.g., [3×3, 64]×4 denotes 4 cascaded convolution layers with 64 filters of size 3×3, S2 denotes stride 2.

Image recognition settings. The network architectures used in the paper are elaborated in Table 9 and Table 10. For CIFAR-10 and CIFAR-100, we use batch size 128. We start with learning rate 0.1, divide it when the performance is saturated. For ImageNet-2012, we use batch size 64 and start with learning rate 0.1. The learning rate is divided by 10 when the performance is saturated, and the training is terminated at 500k iterations. For ResNet-50 on ImageNet-2012, we use exactly the same architecture as [1]. Note that, for all the compared methods, we always use the best possible hyperparameters to make sure that the comparison is fair. The baseline has exactly the same architecture and training settings as the one that CoMHE uses. For both half-space MHE and all the variants of CoMHE in hidden layers, we set the weighting hyperparameter as 1 in all experiments. [12] already shows that MHE type of losses are not senstitive to the weighting hyperparameter. We use $1e-5$ for the orthonormal and orthogonal regularizations (the hyperparameter is obtained via cross-validation). If not otherwise specified, standard ℓ_2 weight decay ($1e-4$) is applied to all the neural network including baselines and the networks that use MHE regularization. Note that, all the neuron weights in the neural networks used in the paper are not normalized (unless otherwise specified), but both MHE and CoMHE will normalize the neuron weights while computing the regularization loss. For all experiments, we use $s=2$ in both MHE and CoMHE. As a result, *CoMHE does not need to modify any component of the original neural networks, and it can simply be viewed as an extra regularization loss that can boost the performance*. All the network architectures are implemented by ourselves, so there might be performance difference between our implementation and the original paper due to some different network and optimization hyperparameters. For example, due to the limitation of computation resources, our batch size for ImageNet-2012 is 64 batch size, which might has some performance loss. However, the network and training settings for the baseline and all the compared regularizations are the same, which ensures the fairness of our experiments. In ImageNet classification, we emphasize that our purpose is to compare the gain brought by different regularizations rather than achieving the state-of-the-art performance. We implement the data augmentation in the ImageNet experiment, following AlexNet [65] and SphereNet [32], so the accuracy may be lower than using the more complicated data augmentation used in original ResNet [1].

Point cloud recognition settings. For all the PointNet and PointNet++ experiments, we exactly follow the same setting in the original papers [62, 63] and their official repositories . Specifically, we combine CoMHE regularization to neurons in all the 1×1 convolution layers before the max pooling layer and the multi-layer perceptron classifier after the max pooling layer. All the regularization is added without changing any components in PointNet. For PointNet experiments, we use point number 1024, batch size 32 and Adam optimizer started with learning rate 0.001, the learning rate will decay by 0.7 every 200k iterations, and the training is terminated at 250 epochs. For PointNet++ experiments, since the MRG (multi-resolution grouping) model is not provided in the official repository, we use the SSG (single scale grouping) model as baseline. Specifically, we use point number 1024, batch size 16 and Adam optimizer started with learning rate 0.001, the learning rate will decay by 0.7 every 200k iterations, and the training is terminated at 251 epochs. For all experiments, we use $s = 1$ in both MHE and CoMHE.

We evaluate on PointNet with T-Net and without T-Net in order to demonstrate that CoMHE is not sensitive to architecture modifications. We follow all the default hyperparameters used in the official released code, and the only difference is that we further combine an additional regularization loss for the neurons in each layer. One can observe that CoMHE consistently performs better than half-space MHE [12].

Besides PointNet, we combine CoMHE to PointNet++ [63] and further show the improvement of generalization introduced by CoMHE is agnostic to the architecture. We evaluate PointNet++ with and without CoMHE on ModelNet-40. Note that, we exactly follow the released code in the official repository where PointNet++ uses the single scale grouping model. Because the original paper [63] uses the multi-resolution grouping model, the baseline performance reported in our paper is not as good as the accuracy reported in the original paper. However, our purpose is to validate the effectiveness of CoMHE, so we only focus on the performance gain. One can observe that CoMHE achieves about 0.5% accuracy gain, while half-space MHE [12] only has about 0.2% accuracy gain.

https://github.com/charlesq34/pointnet
https://github.com/charlesq34/pointnet2

B. Experimental Details and Full Results of Different Hyperspherical Minimization Strategies

B.1. General experimental details

Layer	CNN-3	CNN-9
Conv1.x	[3×3, 32]×1	[3×3, 32]×3
Pool1	2×2 Max Pooling, Stride 2	
Conv2.x	[3×3, 64]×1	[3×3, 64]×3
Pool2	2×2 Max Pooling, Stride 2	
Conv3.x	[3×3, 64]×1	[3×3, 64]×3
Pool3	2×2 Max Pooling, Stride 2	
Fully Connected	64	64

Table 11: Our small plain CNN architectures with different convolutional layers for the illustrative experiment in Fig. 1. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

The training details are the same as the CIFAR-100 experiment described in Appendix A, except that we use different network structure here. All the optimizations of CNNs use the Stochastic gradient descent with momentum 0.9. The number of iteration and learning rate decay exactly follows the CIFAR-100 experiment in Section 6.1.1. The results in Fig. 1 are obtained with the CNN-9 described in Table 11. In order to show that the conclusion obtained using CNN-9 is architecture-agnostic, we also conduct the same experiment on CNN-3. The width of CNN-3 and CNN-9 is smaller than the architectures we used in Section 6.1.1, because the orthogonal training consumes more GPU memory when the size of convolution kernels gets larger. Since the width of CNNs is still the same for all the compared regularizations, it will not affect the validity of our conclusions. For standard, MHE and CoMHE training, the weight decay will be used by default. For rotation training, the weight decay is no longer needed since it does not need to learn the weights for neurons. Note that, all networks (with different regularization) are initialized with the same weights and therefore **have the same hyperspherical energy at the beginning**.

B.2. Details of different training strategies

Standard Training. In standard training, we use the conventional end-to-end training for all the neurons in the CNN via back-propagation. The standard training is the same as the way that baselines are trained in Section 6.

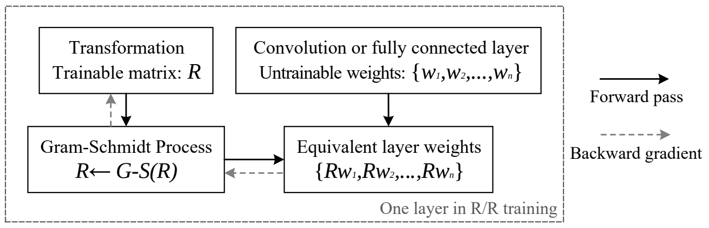

Figure 4: Illustration of one layer in rotation/reflection training.

Rotation Training. We introduce an interesting baseline here for better comparison to our CoMHE. The core of rotation training [66] is to learn an orthogonal matrix for the neurons in the same layer with the weights of these neurons being fixed. Such an orthogonal matrix is learned individually in every layer except the classifier layer (*i.e.*, the final layer that outputs the class logic). The classifier layer is still learned from scratch via back-propagation. Rotation training is a special case of the orthogonal over-parameterized training in [66]. Specifically, we denote N neurons in the i-th (convolution or fully-connected) layer as $\{\boldsymbol{w}_1^{(i)}, \boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{w}_N^{(i)}\} \in \mathbb{R}^d$. After randomly initializing these neuron weights with the method in [67], we will fix $\{\boldsymbol{w}_1^{(i)}, \boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{w}_N^{(i)}\}$ and make them **untrainable** during the entire training procedure. We will learn an orthogonal matrix $\boldsymbol{R}^{(i)} \in \mathbb{R}^{d\times d}$ for the neurons in the i-th layer such that the equivalent neurons become $\{\boldsymbol{R}^{(i)}\boldsymbol{w}_1^{(i)}, \boldsymbol{R}^{(i)}\boldsymbol{w}_2^{(i)}, \cdots, \boldsymbol{R}^{(i)}\boldsymbol{w}_N^{(i)}\}$. The angle between the j-th neuron and k-th neuron is preserved after the rotation/reflection, because we have

$$\cos(\theta_{(\boldsymbol{R}\boldsymbol{w}_j,\boldsymbol{R}\boldsymbol{w}_k)}) = \frac{(\boldsymbol{R}\boldsymbol{w}_j)^\top \boldsymbol{R}\boldsymbol{w}_k}{\|\boldsymbol{R}\boldsymbol{w}_j\| \cdot \|\boldsymbol{R}\boldsymbol{w}_k\|} = \frac{\boldsymbol{w}_j^\top \boldsymbol{w}_k}{\|\boldsymbol{w}_j\| \cdot \|\boldsymbol{w}_k\|} = \cos(\theta_{(\boldsymbol{w}_j,\boldsymbol{w}_k)}). \tag{11}$$

Therefore, rotation training only learns the orthogonal matrices $\{\boldsymbol{R}^{(1)}, \boldsymbol{R}^{(2)}, \cdots, \boldsymbol{R}^{(L)}\}$ for neurons of all the convolution and fully-connected layers except the classifier layer. This training strategy will always keep the hyperspherical energy the same during the training process.

For the implementation of rotation training, we use the Gram-Schmidt process to orthonormalizing all the learnable matrices $\boldsymbol{R}^{(i)}, \forall i$ before multiplying them to the neuron weights. Since the Gram-Schmidt process is differentiable, we can directly insert it to process the learnable matrices $\boldsymbol{R}^{(i)}, \forall i$ in the forward pass. We show an overview of forward and backward pass in one layer of rotation training in Fig. 4 to demonstrate the procedure how we orthonormalize the matrices $\boldsymbol{R}^{(i)}, \forall i$ and apply them to the fixed neuron weights.

MHE Training. MHE training is to train the neurons with both data fitting loss and MHE regularization loss from scratch, following the same procedure in [12].

CoMHE Training Similar to MHE training, CoMHE training is to train the neurons with both data fitting loss and CoMHE regularization loss (including RP-CoMHE and AP-CoMHE) from scratch. Details are given in the main paper.

B.3. Experimental results

Experiments on CIFAR-100 with CNN-9 (BatchNorm) (also shown in Fig. 1). We first conduct experiments on CIFAR-100 with CNN-9 as the backbone architecture. For all the compared methods, we use batch normalization. Note that, the experimental results here are the extended results of Fig. 1. We compute the hyperspherical energy of N neurons using the following definition of half-space hyperspherical energy ($s=1$) [12] (the same as the regularization loss):

$$\boldsymbol{E}=\frac{1}{2N(2N-1)}\sum_{i=1}^{2N}\sum_{j=1,j\neq i}^{2N}\frac{1}{\|\hat{\boldsymbol{w}}_i-\hat{\boldsymbol{w}}_j\|} \tag{12}$$

where $\hat{\boldsymbol{w}}_{N+i}=-\hat{\boldsymbol{w}}_i,\ 0\leq i\leq N$. This is the hyperspherical energy of neurons in one layer. The total hyperspherical energy needs to sum up the energy from all the layers. We show the hyperspherical energy and the accuracy v.s. iteration in Fig. 5.

The results in Fig. 5 shows that rotation training can largely improve the accuracy compared to the baseline, indicating that the hyperspherical energy can characterize the generalization and lower hyperspherical energy leads to better generalization. The rotation training is able to perform similarly to HS-MHE, showing the advantage of low hyperspherical energy. It also shows that the performance of the original MHE (*i.e.*, HS-MHE) can be achieved by a simple rotation/reflection training strategy.

Experiments on CIFAR-100 with CNN-3 (BatchNorm). To show that the same behavior will also happen in different network structure, we conduct experiments on CIFAR-100 with a 3-layer CNN as shown in Table 11. Batch normalization is also used. The results in Fig. 6 confirm that rotation training performs better than the baseline and MHE but still worse than our proposed CoMHE. The results verify that hyperspherical energy is an important generaliability indicator for trained networks.

Experiments on CIFAR-100 with CNN-9 (no BatchNorm). To show that batch normalization does not affect our conclusion, we also conduct experiments on CIFAR-100 without using batch normalization. We use the CNN-9 architecture as shown in Table 11. The results in Fig. 7 show that our conclusion holds even without batch normalization. Hyperspherical energy plays an important role in the network generalization. In general, lower hyperspherical energy leads to better generalization, even though such low hyperspherical energy (in rotation training) is achieved by zero-mean Gaussian initialization.

Experiments on CIFAR-10 with CNN-9 (BatchNorm). To show that the same empirical behavior is consistent in different dataset, we further conduct experiments on CIFAR-10. We use CNN-9 with batch normalization as our backbone architecture. The results in Fig. 8 show that rotation training still performs much better than the baseline and slightly worse than CoMHE. Most interestingly, rotation training can even perform better than the original MHE.

B.4. Conclusion and Discussion

We have extensively tested the hyperspherical energy and accuracy of standard, rotation/reflection, MHE, and CoMHE training under multiple circumstances. The empirical evidences consistently show that hyperspherical energy is of great importance and is able to indicate the potential generalizability of a trained network. Even if we use randomly initialized neurons with low hyperspherical energy, we can still have impressive performance (better than MHE) if proper rotations/reflections of these neurons are learned. Note that rotation/reflection will not change the hyperspherical energy. Therefore, how to effectively minimize the hyperspherical energy is of great significance and is also the central focus of CoMHE.

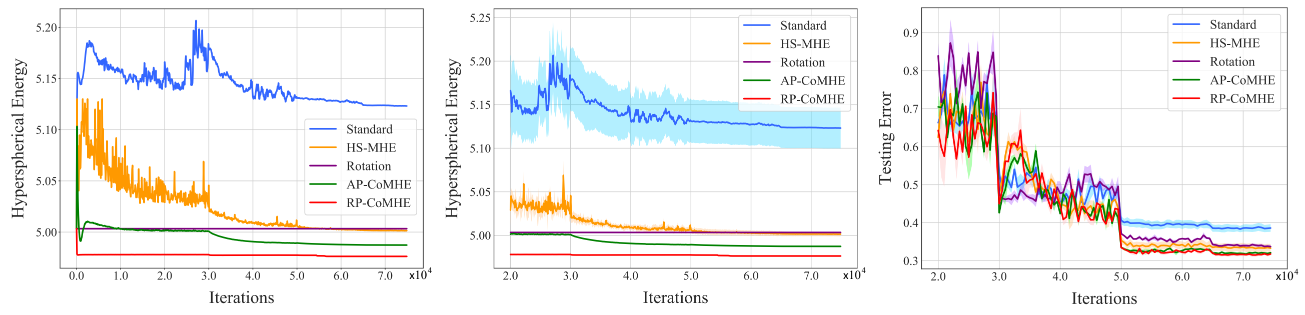

Figure 5: Results on CIFAR-100 with CNN-9 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

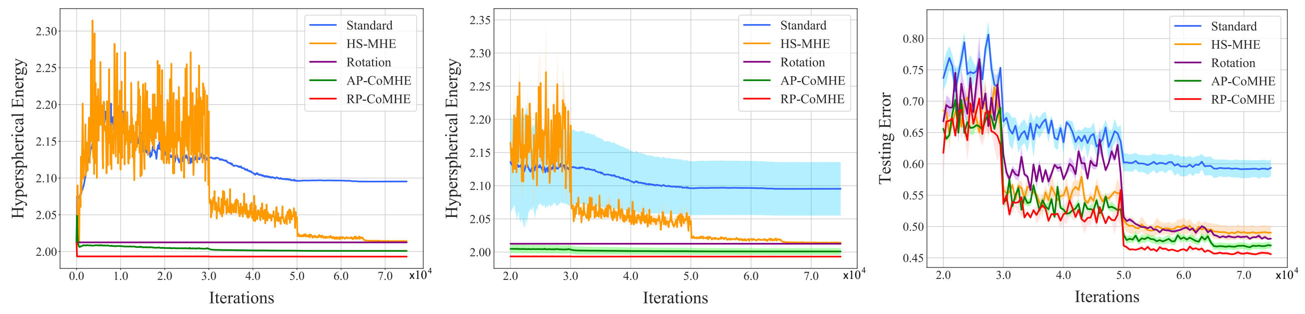

Figure 6: Results on CIFAR-100 with CNN-3 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

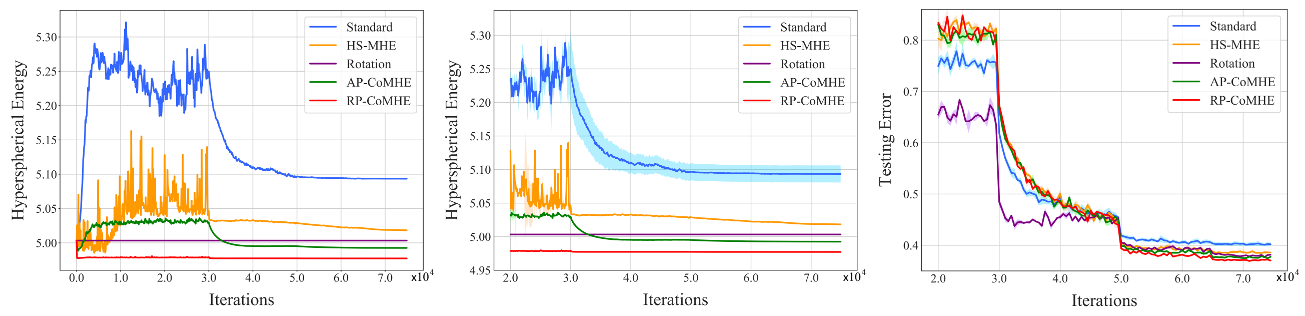

Figure 7: Results on CIFAR-100 with CNN-9 (no BatchNorm is applied). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-100 (with standard deviation).

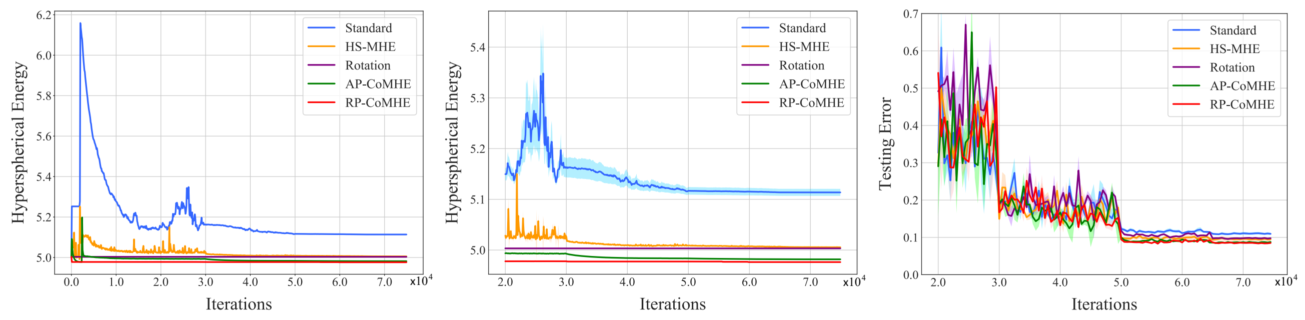

Figure 8: Results on CIFAR-10 with CNN-9 (BatchNorm). Left: hyperspherical energy v.s. iteration during the entire training. Middle: hyperspherical energy v.s. iteration after the 20000-th iterations (with standard deviation). Right: Testing Error on CIFAR-10 (with standard deviation).

C. Two Random Vectors Are Approximately Orthogonal in High Dimensions

We have two random uniform vectors $\frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}$ and $\frac{\boldsymbol{Y}}{\|\boldsymbol{Y}\|}$ where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are normal distributions. Then the inner product of these two independent unit vectors is $\frac{\langle \boldsymbol{X},\boldsymbol{Y}\rangle}{\|\boldsymbol{X}\|\|\boldsymbol{Y}\|}$. When $n \to +\infty$, according to the law of large numbers, we have that $\frac{\boldsymbol{X}}{\sqrt{n}} \to 1$ almost surely. By the central limit theorem, $\frac{\langle \boldsymbol{X},\boldsymbol{Y}\rangle}{\sqrt{n}}$ converges in distribution to a standard one-dimensional normal distribution. Therefore, we have in distribution that

$$\sqrt{n}\cdot\langle \boldsymbol{U},\boldsymbol{V}\rangle \to z \tag{13}$$

where $\boldsymbol{U}=\frac{\boldsymbol{X}}{\|\boldsymbol{X}\|}$, $\boldsymbol{V}=\frac{\boldsymbol{Y}}{\|\boldsymbol{Y}\|}$ and z follows a normal distribution. Then for every $\epsilon > 0$, we have that

$$P(|\langle \boldsymbol{U},\boldsymbol{V}\rangle|) \to 0 \tag{14}$$

which implies that the probability that $\boldsymbol{U}$ and $\boldsymbol{V}$ are nearly orthogonal approaches to 1 when $n \to +\infty$. Similarly, we can also conclude that k independent uniform unit vectors on the hypersphere are nearly orthogonal with very high probability when the dimension becomes higher.

D. Johnson-Lindenstrauss Lemma

Lemma 2 (Johnson-Lindenstrauss Lemma [47, 48])**.** *Let $w_1, w_2 \in \mathbb{R}^d$ be vectors, and $\boldsymbol{P} \in \mathcal{R}^{k\times d}, k < d$ be a random projection matrix with entries i.i.d. drawn from a 0-mean σ-subgaussian distribution. With $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \in \mathbb{R}^k$ being the projected vectors of $\boldsymbol{w}_1, \boldsymbol{w}_2$, then, $\forall\epsilon \in (0,1)$,*

$$(1-\epsilon)\left\|\boldsymbol{w}_1-\boldsymbol{w}_2\right\|^2 k\sigma^2 < \left\|\boldsymbol{P}\boldsymbol{w}_1-\boldsymbol{P}\boldsymbol{w}_2\right\|^2 < (1+\epsilon)\left\|\boldsymbol{w}_1-\boldsymbol{w}_2\right\|^2 k\sigma^2 \tag{15}$$

holds with probability at least $1-2\exp(-\frac{k\epsilon^2}{8})$.

E. Proofs

In the section, we aim to provide the complete proof for self-containedness.

E.1. Lemma 1

We take the expectation of the inner product between projected vectors:

$$\begin{aligned}
\mathbb{E}(\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2\rangle) &= \frac{1}{n}\mathbb{E}\bigg(\sum_{l=1}^{n}\big(\sum_{j=1}^{d} r_{lj}\{\boldsymbol{w}_1\}_j \sum_{i=1}^{d} r_{li}\{\boldsymbol{w}_2\}_i\big)\bigg) \\
&= \frac{1}{n}\sum_{l=1}^{n}\bigg(\sum_{j=1}^{d}\mathbb{E}(r_{lj}^2)\{\boldsymbol{w}_1\}_j\{\boldsymbol{w}_2\}_j + \sum_{j=1}^{d}\mathbb{E}(r_{lj})\{\boldsymbol{w}_1\}_j \cdot \sum_{i\neq j:i=1}^{d}\mathbb{E}(r_{li})\{\boldsymbol{w}_2\}_i\bigg) \\
&= \langle \boldsymbol{w}_1, \boldsymbol{w}_2\rangle
\end{aligned} \tag{16}$$

where $\{\boldsymbol{w}_1\}_i$ is the i-th element of the vector $\boldsymbol{w}_1$, and $\{\boldsymbol{w}_2\}_i$ is the i-th element of the vector $\boldsymbol{w}_2$. From the equation, we see that the lemma is proved. □

E.2. Theorem 1

Before proving the the main theorem, we first show a lemma from [48].

Lemma 3 (Dot Product under Random Projection)**.** *Let $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^d$, $\boldsymbol{P} \in \mathbb{R}^{k\times d}, k < d$ be a random projection matrix having i.i.d. 0-mean subgaussian entries with parameter σ^2, and $\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2$ be the images of $\boldsymbol{w}_1, \boldsymbol{w}_2$ under projection $\boldsymbol{P}$. Then, $\forall\epsilon \in (0,1)$:*

$$\boldsymbol{w}_1^\top\boldsymbol{w}_2 k\sigma^2 - \epsilon k\sigma^2\left\|\boldsymbol{w}_1\right\|\left\|\boldsymbol{w}_2\right\| < (\boldsymbol{P}\boldsymbol{w}_1)^\top\boldsymbol{P}\boldsymbol{w}_2 < \boldsymbol{w}_1^\top\boldsymbol{w}_2 k\sigma^2 + \epsilon k\sigma^2\left\|\boldsymbol{w}_1\right\|\left\|\boldsymbol{w}_2\right\| \tag{17}$$

holds with probability $1-2\exp(-\frac{k\sigma^2}{8})$.

From Lemma 2, we have that

$$\begin{aligned}(1-\epsilon)\left\|\boldsymbol{w}_1\right\|^2 k\sigma^2 < \left\|\boldsymbol{P}\boldsymbol{w}_1\right\|^2 < (1+\epsilon)\left\|\boldsymbol{w}_1\right\|^2 k\sigma^2 \\ (1-\epsilon)\left\|\boldsymbol{w}_2\right\|^2 k\sigma^2 < \left\|\boldsymbol{P}\boldsymbol{w}_2\right\|^2 < (1+\epsilon)\left\|\boldsymbol{w}_2\right\|^2 k\sigma^2\end{aligned} \tag{18}$$

which holds with probability $\left(1-2\exp(-\frac{k\epsilon^2}{8})\right)^2$.

Then we combine Eq. 18 to Lemma 3 and obtain that

$$\frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)})-\epsilon}{1+\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}) < \frac{\cos(\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)})+\epsilon}{1-\epsilon} \tag{19}$$

which holds with probability $\left(1-2\exp(-\frac{k\epsilon^2}{8})\right)^2$. $\theta_{(\boldsymbol{P}\boldsymbol{w}_1,\boldsymbol{P}\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{P}\boldsymbol{w}_1$ and $\boldsymbol{P}\boldsymbol{w}_2$, and $\theta_{(\boldsymbol{w}_1,\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. □

E.3. Theorem 2

Before proving our main theorem, we first show a lemma [49] below:

Lemma 4. *For any $w \in \mathbb{R}^d$, any random Gaussian matrix $\boldsymbol{P} \in \mathbb{R}^{k\times d}$ where $\boldsymbol{P}_{ij} = \frac{1}{\sqrt{n}}r_{ij}$ and $r_{ij}, \forall i,j$ are i.i.d. random variables from $\mathcal{N}(0,1)$, and $\epsilon \in (0,1)$*

$$\Pr\left((1-\epsilon) \le \frac{\left\|\boldsymbol{P}\boldsymbol{w}\right\|^2}{\left\|\boldsymbol{w}\right\|^2} \le (1+\epsilon)\right) \ge 1-2\exp\big(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3})\big) \tag{20}$$

Proof of Lemma 4. From Lemma 1, we have that $\mathbb{E}(\left\|\boldsymbol{P}\boldsymbol{w}\right\|^2) = \left\|\boldsymbol{w}\right\|^2$. Due to 2-stability of the Gaussian distribution, we have that $\sum_{j=1}^d r_{lj}w_j = \left\|\boldsymbol{w}\right\| z_l$ where $z_l \sim \mathcal{N}(0,1)$. As a result, we have that

$$\left\|\boldsymbol{P}\boldsymbol{w}\right\|^2 = \frac{1}{n}\boldsymbol{w}^2\sum_{l=1}^n z_l^2 \tag{21}$$

where $\sum_{l=1}^n z_l^2$ is chi-square distributed with n-degree freedom. Then we apply the standard tail bound of the chi-square distribution and obtain

$$\begin{aligned}\Pr\left(\left\|\boldsymbol{P}\boldsymbol{w}\right\|^2 \le (1-\epsilon)\left\|\boldsymbol{w}^2\right\|\right) &\le \exp\left(\frac{n}{2}\big(1-(1-\epsilon)+\ln(1-\epsilon)\big)\right) \\ &\le \exp(-\frac{n}{4}\epsilon^2)\end{aligned} \tag{22}$$

where the inequality $\ln(1-\epsilon) \le -\epsilon-\frac{\epsilon^2}{2}$ is applied. Similarly, one can have

$$\begin{aligned}\Pr\left(\left\|\boldsymbol{P}\boldsymbol{w}\right\|^2 \le (1+\epsilon)\left\|\boldsymbol{w}^2\right\|\right) &\le \exp\left(\frac{n}{2}\big(1-(1+\epsilon)+\ln(1+\epsilon)\big)\right) \\ &\le \exp(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}))\end{aligned} \tag{23}$$

where the inequality $\ln(1+\epsilon) \le \epsilon-\frac{\epsilon^2}{2}+\frac{\epsilon^3}{3}$ is used. □

From the lemma above, we apply the union bound and have that

$$\begin{aligned}(1-\epsilon) \le \frac{\left\|\boldsymbol{P}\boldsymbol{w}_1\right\|^2}{\left\|\boldsymbol{w}_1\right\|^2} \le (1+\epsilon) \\ (1-\epsilon) \le \frac{\left\|\boldsymbol{P}\boldsymbol{w}_2\right\|^2}{\left\|\boldsymbol{w}_2\right\|^2} \le (1+\epsilon)\end{aligned} \tag{24}$$

which holds with probability at least $1-4\exp(-\frac{n}{2}(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}))$. Using Eq. 24, we can have that

$$\left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right\|^2 \leq \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\sqrt{1-\epsilon}\,\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\sqrt{1-\epsilon}\,\|\boldsymbol{w}_2\|} \right\|^2 \tag{25}$$

From Eq. 24 and the condition that $\boldsymbol{w}_1^\top \boldsymbol{w}_2 > 0$, we further have that

$$\begin{aligned} \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 &\leq \left\| \sqrt{1+\epsilon} - \sqrt{1-\epsilon} \right\|^2 \\ &\leq \left\| \sqrt{1+\epsilon} \left(\frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right) \right\|^2 + \left\| \sqrt{1+\epsilon} - \sqrt{1-\epsilon} \right\|^2 \end{aligned} \tag{26}$$

Then we apply Lemma 4 to the vector $(\frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|})$ and see that

$$(1-\epsilon) \left\| \frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \leq \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \leq (1+\epsilon) \left\| \frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2 \tag{27}$$

which holds with probability $1 - 2\exp\big(-\frac{n}{2}(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})\big)$. Then we have that

$$\begin{aligned} \frac{\langle \boldsymbol{w}_1, \boldsymbol{w}_2 \rangle}{\|\boldsymbol{w}_1\|\,\|\boldsymbol{w}_2\|} &= 1 - \frac{1}{2} \left\| \frac{\boldsymbol{w}_1}{\|\boldsymbol{w}_1\|} - \frac{\boldsymbol{w}_2}{\|\boldsymbol{w}_2\|} \right\|^2, \\ \frac{\langle \boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2 \rangle}{\|\boldsymbol{P}\boldsymbol{w}_1\|\,\|\boldsymbol{P}\boldsymbol{w}_2\|} &= 1 - \frac{1}{2} \left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right\|^2. \end{aligned} \tag{28}$$

From Eq. 25, Eq. 26 and Eq. 27, we can learn that $\left\| \frac{\boldsymbol{P}\boldsymbol{w}_1}{\|\boldsymbol{P}\boldsymbol{w}_1\|} - \frac{\boldsymbol{P}\boldsymbol{w}_2}{\|\boldsymbol{P}\boldsymbol{w}_2\|} \right\|^2$ is bounded below and above. Further combining Eq. 28, we have that

$$\frac{1+\epsilon}{1-\epsilon} \cos(\theta_{(\boldsymbol{w}_1, \boldsymbol{w}_2)}) - \frac{2\epsilon}{1-\epsilon} < \cos(\theta_{(\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2)}) < \frac{1-\epsilon}{1+\epsilon} \cos(\theta_{(\boldsymbol{w}_1, \boldsymbol{w}_2)}) + \frac{1+2\epsilon}{1+\epsilon} - \frac{\sqrt{(1-\epsilon^2)}}{1+\epsilon} \tag{29}$$

where $\theta_{(\boldsymbol{P}\boldsymbol{w}_1, \boldsymbol{P}\boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{P}\boldsymbol{w}_1$ and $\boldsymbol{P}\boldsymbol{w}_2$, and $\theta_{(\boldsymbol{w}_1, \boldsymbol{w}_2)}$ denotes the angle between $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. □

E.4. Theorem 3

We first introduce a lemma before proving the theorem.

Lemma 5. *[Direct Result from [68]] Let $\mathcal{H}$ be a separable Hilbert space, and let μ be a non-degenerate Gaussian measure on $\mathcal{H}$. Let P, Q be Borel probability measures on $\mathcal{H}$. Assume that:*

- *The abosolute moments $m_n := \int \|x\|^n \, dP(x)$ are finite and satisfy $\sum_{n\geq 1} m_n^{\frac{-1}{n}} = \infty$;*
- *The set $\varepsilon(P, Q) := \{x \in \mathcal{H} : P_{\langle x \rangle} = Q_{\langle x \rangle}\}$, where $\langle x \rangle$ denotes the one-dimensional subspace spanned by x, is of positive μ-measure.*

Then we have $P = Q$.

If we consider $\boldsymbol{w} \in \mathbb{R}^d$ as a bounded variable, and without loss of generality, we assume that $\boldsymbol{p} = \boldsymbol{z}/\|\boldsymbol{z}\|$ where $\boldsymbol{z}$ is a Gaussian distribution, and then the condition on the moments of $\boldsymbol{w}$ in Lemma 5 holds. Then with the following lemma, we can easily have the desired result.

□

F. Bilateral Projection for CoMHE

In this section, we consider bilateral projection for CoMHE (BP-CoMHE) as an extension to the main paper. If we view the neurons in one layer as a matrix $\boldsymbol{W}=\{\boldsymbol{w}_1,\cdots,\boldsymbol{w}_n\}\in\mathbb{R}^{m\times n}$ where m is the dimension of neurons and n is the number of neurons, then the projection mainly considered throughout the paper is to left-multiply a projection matrix $\boldsymbol{P}_1\in\mathbb{R}^{r\times m}$ to $\boldsymbol{W}$. In fact, we can further reduce the number of neurons by right-multiplying an additional projection matrix $\boldsymbol{P}_2\in\mathbb{R}^{n\times r}$ to $\boldsymbol{W}$. Specifically, we denote that

$$\boldsymbol{Y}_1=\boldsymbol{P}_1\boldsymbol{W}\in\mathbb{R}^{r\times n},\quad \boldsymbol{Y}_2=\boldsymbol{W}\boldsymbol{P}_2\in\mathbb{R}^{m\times r} \tag{30}$$

BP-CoMHE. The first variant of BP-CoMHE is to apply the MHE regularization separately to column vectors of $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, and the learning objective is given by

$$\min_{\boldsymbol{W}}\boldsymbol{E}_s(\hat{\boldsymbol{y}}_i^{(1)}|_{i=1}^n)+\boldsymbol{E}_s(\hat{\boldsymbol{y}}_i^{(2)}|_{i=1}^r) \tag{31}$$

where we denote that

$$\begin{aligned}
&\boldsymbol{Y}_1=\{\boldsymbol{y}_1^{(1)},\cdots,\boldsymbol{y}_n^{(1)}\}\in\mathbb{R}^{r\times n},\\
&\boldsymbol{Y}_2=\{\boldsymbol{y}_1^{(2)},\cdots,\boldsymbol{y}_r^{(2)}\}\in\mathbb{R}^{m\times r},\\
&\hat{\boldsymbol{Y}}_1=\{\hat{\boldsymbol{y}}_1^{(1)}=\frac{\boldsymbol{y}_1^{(1)}}{\|\boldsymbol{y}_1^{(1)}\|},\cdots,\hat{\boldsymbol{y}}_n^{(1)}=\frac{\boldsymbol{y}_n^{(1)}}{\|\boldsymbol{y}_n^{(1)}\|}\}\in\mathbb{R}^{r\times n},\\
&\hat{\boldsymbol{Y}}_2=\{\hat{\boldsymbol{y}}_1^{(2)}=\frac{\boldsymbol{y}_1^{(2)}}{\|\boldsymbol{y}_1^{(2)}\|},\cdots,\hat{\boldsymbol{y}}_r^{(2)}=\frac{\boldsymbol{y}_r^{(2)}}{\|\boldsymbol{y}_r^{(2)}\|}\}\in\mathbb{R}^{m\times r}.
\end{aligned} \tag{32}$$

in which we have that $\hat{\boldsymbol{y}}_i^{(1)}\in\mathbb{R}^{r\times 1}$ and $\hat{\boldsymbol{y}}_i^{(1)}\in\mathbb{R}^{m\times 1}$ are two column vectors of $\hat{\boldsymbol{Y}}_1$ and $\hat{\boldsymbol{Y}}_2$, respectively. The final neurons obtained for the neural network are still $\boldsymbol{W}$. For generating the projection matrices $\boldsymbol{P}_1,\boldsymbol{P}_2$, we simply use random projection and re-initialize the random matrices every certain number of iterations.

Low-Rank BP-CoMHE. More interestingly, we can also approximate $\boldsymbol{W}$ with a low-rank factorization [56] given as follows:

$$\tilde{\boldsymbol{W}}=\boldsymbol{Y}_2(\boldsymbol{P}_1\boldsymbol{Y}_2)^{-1}\boldsymbol{Y}_1\in\mathbb{R}^{m\times n} \tag{33}$$

which inspires us to directly use two set of parameters $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ to represent the equivalent neurons $\tilde{\boldsymbol{W}}$ and apply the MHE regularization separately to their column vectors (similar to the previous BP-CoMHE). Essentially, we learn the matrices $\boldsymbol{Y}_1,\boldsymbol{Y}_2$ directly via back-propagation. The projection matrix $\boldsymbol{P}_1$ is initialized as a random matrix and stays constant during the training. Different from the former case, we will not use $\boldsymbol{W}$ as the final neurons in the neural network. Instead, we will use $\tilde{\boldsymbol{W}}$ as the final neurons. The number of learnable parameters in total is $mr+nr$, which is significantly lower than the original BP-CoMHE parameterization (*i.e.*, mn) if we choose r to be much smaller than both m and n.

G. More Discussion on the Effectiveness of CoMHE

G.1. Full results of Figure 2 in the main paper

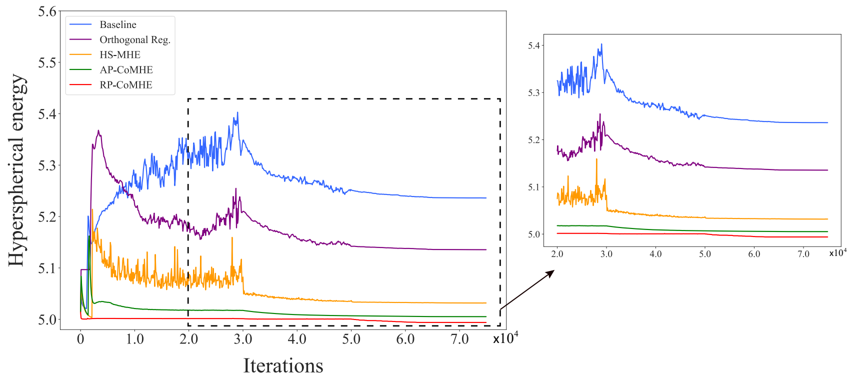

Figure 9: Hyperspherical energy during the entire training. Note that, all networks are initialized with the same weights and therefore have the same hyperspherical energy at the beginning. Note that, “Orthogonal Reg.” denotes the orthogonal regularization (use orthogonal constraint to regularize the neurons), which is dramatically different from the rotation/reflection training that is mentioned above and learns orthogonal matrices for neurons.

Fig. 9 shows the entire training dynamics (from initialization to the end of training) of the hyperspherical energy of baseline CNN and CNN regularized by orthogonal regularization, HS-MHE, AP-CoMHE and RP-CoMHE. All the networks use exactly the same initialized weights to ensure the hyperspherical energy is the same at the beginning. One can observe that the hyperspherical energy is actually very low for the initialized weights. This is because the initialized weights follows Gaussian distribution and the hyperspherical energy is computed with normalized weights. The normalized weights (sampled from Gaussian distribution) follows the uniform distribution on the hypersphere (see Theorem 1 in [66]), which can obtain the lowest hyperspherical distribution in expectation. However, when the weights of the neural network start to fit the data and minimize the data approximation loss, the neuron weights no longer follow the hyperspherical uniform distribution. Therefore the hyperspherical energy will quickly get large. This is when MHE and CoMHE are useful. From Fig. 9, one can see that without any regularization on hyperspherical energy, the hyperspherical energy of the baseline network gets extremely large at the beginning and then slowly decreases as the training continues. However, the final hyperspherical energy of the baseline network is still way higher than the CNNs regularized by MHE and CoMHE. Notice that, the orthogonality-regularized CNN also obtain high hyperspherical energy at the end (similar to the baseline network). In contrast to MHE, we can observe that CoMHE can effectively minimize the hyperspherical energy and RP-CoMHE achieves significantly lower hyperspherical energy in the end, which well verifies the superiority of the proposed CoMHE.

G.2. Hyperspherical energy dynamics in individual layers

To demonstrate the hyperspherical energy dynamics in individual layers, we show the hyperspherical energy v.s. iteration in every layer of CNN-9 (as specified in Table 9) in Fig. 10. Since the last fully-connected layer (*i.e.*, classifier layer) is learned from scratch (no rotation training is applied), we do not plot its hyperspherical energy. From the results in Fig. 10, we can observe that CoMHE can more effectively minimize the hyperspherical energy in every layer, and RP-CoMHE performs the best in terms of the hyperspherical energy minimization.

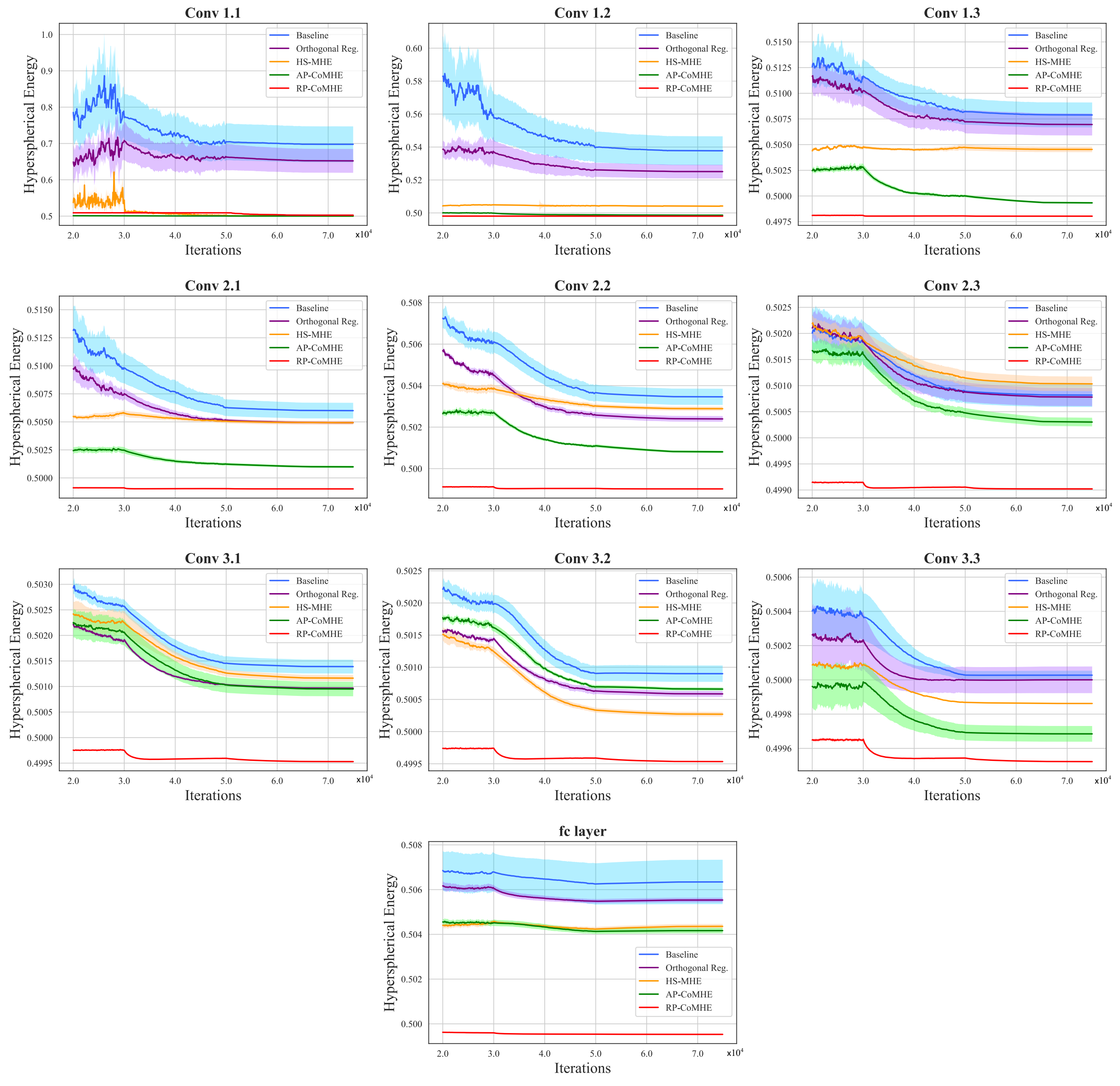

Figure 10: Hyperspherical energy of every layer (Conv1.1, Conv1.2, Conv1.3, Conv2.1, Conv2.2, Conv2.3, Conv3.1, Conv3.2, Conv3.3, fc1) after the 20000-th iteration. Note that, all networks are initialized with the same weights and therefore have the same hyperspherical energy at the beginning.

H. Additional Exploratory Experiments

Frequency of re-initialization in RP-CoMHE. In RP-CoMHE, we need to re-initialize the random projections every certain number of iterations to avoid trivial solutions caused by bad initialization. Here, we test how the frequency of re-initialization will affect the accuracy on CIFAR-100, with the projection dimension being 30 and the number of projection being 20. The iteration number being ∞ in Table 12 represents that the random projection is fixed throughout the training once it is initialized. The results shows the performance is not very sensitive to the frequency of re-initialization, but we cannot fix the random projection during training as it may cause trivial solutions and hurt the performance.

# Iterations	1	200	1000	∞
RP-CoMHE	**24.6**	24.84	24.62	26.09

Table 12: Error of different # iterations for re-initialization.

Naively learning projection basis from training data. We study the case where we enable the back-propagation gradient to flow back to the projection basis. That is to say, the model learns the projection basis naively using training data. We find that naively learning the projection basis yields much worse performance (26.5%), compared to RP-CoMHE (24.6%). It is even worse than our baseline half-space MHE (25.96%). The results show that naively learning projection basis from training data leads to inferior performance. Allowing the projection basis to be updated according to the training data could undermine the strength of CoMHE regularization imposed on the neurons.

Shared projection basis. We take RP-CoMHE as an example to empirically verify the advantages of shared projection basis across different layers. We set the projection dimension to 20 and the number of projections to 30. The plain CNN-9 is used as baseline. Specifically for shared projection basis, we share the random projection basis in Conv1.x, Conv2.x and Conv3.x separately. The shared projection yields 24.6% error rate. For independent projection basis, we use separated projection basis for different layers and only obtain 26.05% error rate. The results show that using shared random projection basis for neurons of the same dimensionality improves the network generalization while saving parameters. Note that, all the other experiments use shared projection basis by default.

I. Training Runtime Comparison

We also provide runtime comparison for all the proposed CoMHE. We use the plain CNN-9 for all the methods in this experiment. For RP, we set the projection dimension to 30 and the number of projection to 5. For AP, the number of projection is 1 and the projection dimension is set to 30. This hyperparameter setting for CoMHE can achieve the best testing accuracy on CIFAR-100. The results in Table 13 are computed using the total runtime of runing 100 iterations. We can see that the runtime of RP-CoMHE, AP-CoMHE and Adv-CoMHE is comparable to HS-MHE and the baseline. Without any code optimization, RP-CoMHE is 36% slower than the baseline and 18% slower than the HS-MHE, and AP-CoMHE is 34% slower than the baseline and 17% slower than the HS-MHE. Note that, although CoMHE is relatively slower in terms of training runtime, CoMHE will not affect the testing runtime of a trained model. That is to say, CoMHE-regularized CNN has the same inference speed with its baseline CNN counterpart. In fact, as long as the training time of CoMHE-regularized CNNs is not geometrically larger than the standard CNN, such computational cost is neglectable in practice and practitioners usually care more about the inference time rather than the training time (CoMHE will not affect the inference time).

Method	Runtime (s)
Baseline	5.61
HS-MHE	6.46
RP-CoMHE	7.62
AP-CoMHE	7.48
Adv-CoMHE	6.37
Group CoMHE	11.12

Table 13: Training Runtime (s / 100 iterations) comparison on CIFAR-100.

J. Experiments on Graph Convolutional Networks

We also use CoMHE to improve graph convolutonal networks (GCN) [69] for node classification in a graph. We use the official code from [69], so the experimental setting and hyperparameter setup are exactly the same as [69]. The only difference is that we apply an additional MHE or CoMHE to the weight matrix. Specifically, the graph convolution network uses the following forward model:

$$\boldsymbol{Z} = \text{Softmax}\big(\hat{\boldsymbol{A}} \cdot \text{ReLU}(\hat{\boldsymbol{A}} \cdot \boldsymbol{X} \cdot \boldsymbol{W}_0) \cdot \boldsymbol{W}_1\big) \tag{34}$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{\frac{1}{2}}$. We note that $\boldsymbol{A}$ is the adjacency matrix of the graph, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ ($\boldsymbol{I}$ is an identity matrix), and $\tilde{\boldsymbol{D}} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the feature matrix of n nodes in the graph (feature dimension is d). $\boldsymbol{W}_1$ is the weights of the classifiers. $\boldsymbol{W}_0$ is the weight matrix of size $d \times h$ where h is the dimension of the hidden space. We view every column of $\boldsymbol{W}_0$ as a neuron, and therefore, there will be h neurons in total. We simply apply MHE or CoMHE to regularize these h neurons. The experimental results are given in Table 14. We can see from the results that the CoMHE-regularized GCN can consistently outperform the MHE-regularized GCN and the GCN baseline. We use exactly the same code as in the official repository, and the only difference is the regularization on $\boldsymbol{W}_0$. We emphasize that CoMHE will not change the inference speed of GCN, so this $1\% - 2\%$ performance gain is more like a "free lunch".

Method	Citeseer	Cora	Pubmed
GCN Baseline	70.3	81.3	79.0
HS-MHE [12]	71.5	82.0	79.0
RP-CoMHE	**72.1**	**82.7**	**79.5**
AP-CoMHE	72.0	82.6	**79.5**

Table 14: Classification accuracy (%) of GCN with different hyperspherical energy regularization.

The code is available at https://github.com/tkipf/gcn.