Spherization Layer: Representation Using Only Angles

Hoyong Kim
Artificial Intelligence Graduate School
Gwangju Institute of Science and Technology
hoyong.kim.21@gm.gist.ac.kr

Kangil Kim*
Artificial Intelligence Graduate School
Gwangju Institute of Science and Technology
kangil.kim.01@gmail.com

Abstract

In neural network literature, angular similarity between feature vectors is frequently used for interpreting or re-using learned representations. However, the inner product in neural networks partially disperses information over the scales and angles of the involved input vectors and weight vectors. Therefore, when using only angular similarity on representations trained with the inner product, information loss occurs in downstream methods, which limits their performance. In this paper, we proposed the *spherization layer* to represent all information on angular similarity. The layer 1) maps the pre-activations of input vectors into the specific range of angles, 2) converts the angular coordinates of the vectors to Cartesian coordinates with an additional dimension, and 3) trains decision boundaries from hyperplanes, without bias parameters, passing through the origin. This approach guarantees that representation learning always occurs on the hyperspherical surface without the loss of any information unlike other projection-based methods. Furthermore, this method can be applied to any network by replacing an existing layer. We validate the functional correctness of the proposed method in a toy task, retention ability in well-known image classification tasks, and effectiveness in word analogy test and few-shot learning. Code is publicly available at https://github.com/GIST-IRR/spherization_layer

1 Introduction

The inner product is a key element constituting layers in deep neural networks with a nonlinear activation function. The inner product with the Euclidean norms and the angle, that is, $\|\mathbf{w}_i\|\|\mathbf{x}_j\| \cos \boldsymbol{\theta}_{ij}$, has been analyzed in terms of the norms $\|\mathbf{w}_i\|\|\mathbf{x}_j\|$ and the angle $\cos \boldsymbol{\theta}_{ij}$, independently [14, 15, 37]. All factors of the inner product learn distinct information. Therefore, using only one factor results in information loss when re-using or understanding the information in downstream tasks. This problem is termed as *dispersion problem*. The angular similarity between features is frequently used. However, this technique causes the dispersion problem in advanced methods in neural network literature such as decoupled network [15], representation learning [6, 7, 23, 24, 34], regularization [35, 36], zero-shot learning [26], and generative model [3, 27]. To mitigate the dispersion problem, numerous angle-based learning approaches [1, 12, 13, 16, 18, 17, 35, 36, 38] have been proposed. However, these studies are based on projection onto the hyperspherical surface. In this projection method, distinction by the scale of features is ignored. Therefore, information loss occurs.

We proposed the *spherization layer* as an explicit solution for the dispersion to completely eliminate the interference of the norms in training without drawbacks. This layer is used to locate all representations onto a constrained region on the hyperspherical surface and train hyperplanes passing through the origin to learn representations with only the angles.

*corresponding author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

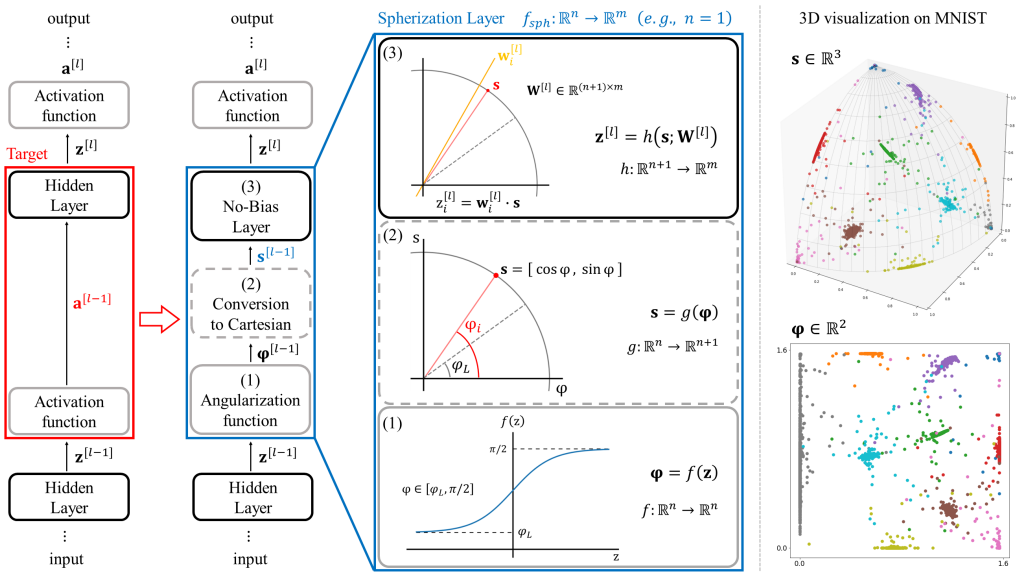


Figure 1: Overview of *spherization layer*. The red box indicates the existing fully connected layer to be replaced with the spherization layer in the blue box. After replacement, the pre-activations from $(l-1)$th layer are passed to the angularization function, not activation function. These pre-activations are converted to angular coordinates in (1). Through (2), spherized representations are located on the $(n+1)$-spherical surface. Finally, a hidden layer without bias parameters is trained on these spherized representations for using only the angles in (3). These relations of the input and output of the stage (1), (2), and (3) are illustrated as graphs, which are clearly defined in Eq. 3, Eq. 6, and Eq. 9, respectively. The right side displays how to generate angular coordinates and convert them to Cartesian coordinates in a 3-dimensional space on MNIST

The spherization layer consists of three main components: *Angularization function* that converts the pre-activations from the previous hidden layer to angles; *Conversion* from spherical coordinates to Cartesian coordinates located on the hyperspherical surface; *No-bias layer*, a hidden layer without bias parameters, that determines decision boundaries by using only the angles. Figure 1 illustrates the design of the spherization layer. Through *spherization*, representations are located on the hyperspherical surface and the effect of the norms in representation learning is completely eliminated. Therefore, neural networks are enforced to express all representations differently by using only the angles.

We experimentally verified the functional correctness of the spherization layer in a toy task and its applicability to feedforward and convolutional neural networks by evaluating performance on image classification tasks. The results reveal that the training ability of original networks is preserved after applying the spherization layer. Furthermore, we analyzed the sensitivity to width and depth, the effect of projection in the spherization layer, and the influence of the spherization layer on the gradient flows in training. Finally, we investigated the effect of the proposed method in downstream tasks through visualization, word analogy test, and few-shot learning.

In summary, our contributions are three-fold:

- To address the dispersion problem, we propose the *spherization layer* to represent all feature vectors on the hyperspherical surface and learn the representations with only the angles.
- We validate the wide-applicability and scalability of the spherization layer without any loss of performance through experiments on various well-known networks.
- We empirically show that the spherization layer can be used in many applications in which angular similarity is a critical metric.

2 Background

Conversion Spherical to the Cartesian Coordinate System In most neural networks, all input samples on the n-dimensional space are represented as Cartesian coordinates, in which i-th column value denotes the distance from the origin along the i-th axis, and neural networks train them by using neurons in which the inner product between the input and weight vector occurred. The pre-activations from neurons are used to determine whether neurons should be activated by using the following activation function. In this process, the pre-activations are calculated by using weight and bias parameters, represented as Cartesian coordinates. These Cartesian coordinates can be converted from spherical coordinates. Given a vector $\mathbf{s}$ represented as Cartesian coordinates on the n-dimensional space, $\mathbf{s}$ can be defined as spherical coordinates $\boldsymbol{\theta} = [r, \boldsymbol{\varphi}]$, composed of a radial coordinate r and n-1 angular coordinates $\boldsymbol{\varphi} = [\varphi_1, \varphi_2, ..., \varphi_{n-1}]$. In this case, the k-th axis of $\mathbf{s}$ can be computed from $\boldsymbol{\theta}$ with Eq. 1

$$\mathbf{s} \;=\; [r\cos\varphi_1, \cdots, r\cos\varphi_k \textstyle\prod_{i=1}^{k-1}\sin\varphi_i, \cdots, r\prod_{i=1}^{n-1}\sin\varphi_i] \tag{1}$$

, where $s_{1<k<n} = r\cos\varphi_k \prod_{i=1}^{k-1}\sin\varphi_i$.

Generally, the spherical coordinate system is a 3-dimensional version of the polar coordinate system. However, the spherical coordinate system in the followings indicates all n-dimensional versions of the polar coordinate system, where n is greater than or equal to 2.

3 Spherization Layer

We proposed the *spherization layer* as shown in Figure 1 to locate feature vectors on the hyperspherical surface without any loss of information and learn hyperplanes by using only the angles. In this method, a layer of an original network is selected to learn representations by angular similarity. Next, the layer is replaced to a *spherization layer* through three sequential stages: *Angularization*, *Conversion to Cartesian*, and *No-bias training* notated as f, g, and h functions, respectively. The final form of spherization layer f_{sph} is expressed as Eq. 2.

$$f_{sph} \;=\; (h \circ g \circ f)\,, \quad f_{sph} : \mathbb{R}^n \rightarrow \mathbb{R}^m \tag{2}$$

After training with the layer, we can obtain the representations on the $(n+1)$-spherical surface, namely spherized representations, as the outputs of $(g \circ f)$ operations.

The common goal of all stages is to preserve the training ability of the original network while the spherization layer trains all information on the $(n+1)$-spherical surface. The angularization locates all pre-activations on the safe spherical surface. The conversion to Cartesian results in the generation of compatible representations to the ordinary layer. No-bias training enforces training by only the angles. In the followings, we elaborate each stages more detail and only annotate the $(l-2)$th and (l)th layer as $[l-2]$ and $[l]$, respectively, for the simplicity.

3.1 Angularization

$$\boldsymbol{\varphi} \;=\; f(\mathbf{z}), \quad f : \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{3}$$

Angularization is the stage to map a pre-activation vector $\mathbf{z}$, passed from $(l-1)$th layer, to angular coordinates $\boldsymbol{\varphi}$. The n indicates the dimension of the pre-activation vector. The role of this stage is to configure the shape of the mapped region on the $(n+1)$-spherical surface for resolving training and computational difficulty.

Angularization f is implemented by applying the following element-wise function to all coordinates of $\mathbf{z}$ as an activation function, and f is illustrated as Eq. 4.

$$f(\mathbf{z}) \;=\; \left(\tfrac{\pi}{2} - \varphi_L\right) \cdot \sigma(\alpha \cdot \mathbf{z}) + \varphi_L \tag{4}$$

, where the terms and form are used by three following motivations.

Converting Pre-Activation to the Angular Coordinate The first step is to convert the input vector into angular coordinates. To ensure the conversion as bijective mapping, we restrict the range of the function as $[0, \frac{\pi}{2}]$. To allow unrestricted input representations on the real-valued domain, the sigmoid function $\sigma(\cdot)$ is used with weight $\frac{\pi}{2}$ for the range setting. As the sigmoid function used,

the input vector should be pre-activations, not activations because when activations from ReLU or another sigmoid are passed to the angularization, inefficient use of the spherical surface or gradient amplification, respectively, may occur. After converting pre-activations to angular coordinates, the representations on the hyperspherical surface in the same range were located by setting a consistent radius over all inputs. This radius scale is controlled in the conversion-to-Cartesian stage.

Tailoring Angular Representation Space In the conversion from angular to Cartesian coordinates, the last coordinate can be an extremely small value because trigonometric values in [0, 1] are multiplied many times. This scale descent can map all values in the axis to only a single value by the limit of the floating point data type. To reduce this effect in the angularization, we introduce a lower bound φ_L of angles to guarantee distinguishable values in its corresponding converted Cartesian coordinates as the following equation (Eq. 5):

$$\varphi_L \;=\; \sin^{-1}\left(\delta^{1/n}\right) \tag{5}$$

, where δ is a minimal trigonometric value to guarantee the distinguishable representations. The details are presented in Appendix A. We set δ to the empirically obtained proper value 10^{-6}, for all experiments.

Scaling Pre-Activations In angularization, the activations are concentrated onto the small region because of the lower bound. This concentration renders training difficult with the decrease in the variance. To reduce the effect, we set a learnable parameter α as a weight of $\mathbf{z}$, which controls the variance of $\mathbf{z}$. Using this scale factor, the generated angular representations become abundant.

3.2 Conversion-to-Cartesian

$$\mathbf{s} \;=\; g(\boldsymbol{\varphi}), \quad g : \mathbb{R}^n \rightarrow \mathbb{R}^{n+1} \tag{6}$$

In the *Conversion-to-Cartesian* stage, the angular coordinates of the previous stage are converted to Cartesian coordinates on the $(n+1)$-spherical surface. This conversion ensures the consistency between the output of angularization (polar coordinate system) and the input of the following no-bias layer (Cartesian coordinate system), and enables the layer to be trained in the same way as general neural networks. Furthermore, an additional dimension makes the spherization layer have enough capacity to be compatible with the ordinary layer. This implementation is based on Eq. 1 with the modified range of angles as the following equation (Eq. 7).

$$g(\boldsymbol{\varphi}) \;=\; [r\cos\varphi_1, \cdots, r\cos\varphi_k \textstyle\prod_{i=1}^{k-1}\sin\varphi_i, \cdots, r\prod_{i=1}^{n}\sin\varphi_i], \quad \varphi_i \in \left[\varphi_L, \frac{\pi}{2}\right] \tag{7}$$

Calculation Trick Implementation of Eq. 7 as a tensor operation requires the trick defined in the following equation (Eq. 8):

$$\begin{aligned} \boldsymbol{\phi} &= \mathbf{W}_{\varphi}^{\intercal}\boldsymbol{\varphi} \\ \mathbf{s} &= r \cdot \exp\left(\mathbf{W}_{\phi}^{\intercal}\ln\left(\sin\boldsymbol{\phi}\right) + \ln\left(\cos\left(\boldsymbol{\phi} + \mathbf{b}_{\phi}\right)\right)\right) \end{aligned} \tag{8}$$

, where $\boldsymbol{\phi}$ is a dimension-expended vector in $\mathbb{R}^{n+1}$ and $\phi_{n+1} = \phi_n$, and r is a constant to control radius. Here, $\mathbf{W}_{\varphi}$, $\mathbf{W}_{\phi}$, and $\mathbf{b}_{\phi}$ are constant matrices and vector in $\mathbb{R}^{n\times(n+1)}$, $\mathbb{R}^{(n+1)\times(n+1)}$, and $\mathbb{R}^{(n+1)}$, respectively. $\mathbf{W}_{\varphi} = [\mathbf{I}_n; \mathbf{v}]$, where $\mathbf{v} = [\mathbf{0}; 1]^{\intercal} \in \mathbb{R}^n$, is used for matching the dimension between $\boldsymbol{\varphi}$ and $\mathbf{s}$. The other constants are used for calculating logarithm trigonometric values from expanded angular coordinates in the way of matrix multiplication, where $\mathbf{W}_{\phi}$ is an upper triangular matrix in which all diagonals are zero and $(\mathbf{W}_{\phi})_{n,n+1}$ is also zero, and $\mathbf{b}_{\phi} = [\mathbf{0}; -\frac{\pi}{2}]^{\intercal}$. See Appendix B for detail hyperparameters and process.

3.3 No-bias Training

$$\mathbf{z}^{[l]} \;=\; h(\mathbf{s}), \quad h : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^m \tag{9}$$

Through angularization and conversion-to-Cartesian, all representations are located on the $(n+1)$-spherical surface. In this case, any update to the representations is determined by the change of angular similarity. However, hyperplanes on the $(n+1)$-dimensional space from the next layer may not use only the angular similarity, which may assign semantic information to the Euclidean norm of the spherized representation. To synchronize these parameters, we used *no-bias training* using only the weight parameters $\mathbf{W}^{[l]}$ as illustrated in Eq. 10.

$$\mathbf{z}^{[l]} \;=\; \mathbf{W}^{[l]\intercal}\,\mathbf{s} \tag{10}$$

Effect of No-Bias on Training In the ordinary layer, the problem of no-bias is that hyperplanes passing through the origin cannot be shifted to another parallel hyperplanes. However, the problem disappears when all feature vectors are located on the $(n+1)$-spherical surface because the decision boundary can be shifted by only the angle changes of $(n+1)$-dimensional hyperplanes even though they pass through the origin.

3.4 Optimization with Overall Process

A training loss L is calculated in the same manner as the original network. The gradient of L for the spherization layer is calculated by multiplying the following partial derivative $\frac{\partial L}{\partial \mathbf{W}}$ to the original backpropagation step from the $(l-2)$th to (l)th layer, as illustrated in Eq. 11. The first term $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ is calculated by the subsequent layers in the same way of the original network.

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}} &= \frac{\partial L}{\partial \mathbf{z}^{[l]}} \cdot \frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{s}} \cdot \frac{\partial \mathbf{s}}{\partial \boldsymbol{\varphi}} \cdot \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}} \\
\frac{\partial \mathbf{z}}{\partial \mathbf{W}} &= \mathbf{a}^{[l-2]} \\
\frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{z}} &= \left(\frac{\pi}{2} - \varphi_L\right) \cdot \alpha \cdot \sigma'(\alpha \cdot \mathbf{z}) \\
\frac{\partial \mathbf{s}}{\partial \boldsymbol{\varphi}} &= [-r \sin \varphi_1, \cdots, -r \prod_{i=1}^{k} \sin \varphi_i, \cdots, r \cos \varphi_n \prod_{i=1}^{n-1} \sin \varphi_i] \\
\frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{s}} &= \mathbf{W}^{[l]}
\end{aligned} \tag{11}$$

4 Experiments

First, we define two terms for simplicity on indicating two networks: one is the network before applying the spherization layer, called *original network*, and the other is the network after substituting an ordinary layer with the spherization layer, called *spherized network*. In the followings, we used these two terms consistently. All experiments were performed five times with random seeds and their training and test accuracy were evaluated except word analogy test and few-shot learning. The mean μ and standard deviation σ of accuracy are represented as $\mu \pm \sigma$ in each table.

4.1 Functional Correctness Test on a Toy Task

Experimental Settings We verified the spherization layer for learning decision boundaries on a simple binary classification task. We set up the simple binary classification task: given $(\mathbf{x}_i, y_i)$ pairs, where $\mathbf{x}_i$ is the i-th input sample in $\mathbb{R}^2$ and $y_i \in \{0, 1\}$ is the label of $\mathbf{x}_i$, we randomly generated 100 input samples located around (0, 0) for the label 0, and the other 100 samples for the label 1 around (1, 1), as shown in Figure 2. We set a 2-layer neural network as the original network, and trained it with the softmax function, cross-entropy, and SGD at a learning rate of 0.01. We applied the proposed method to the original network by replacing the last fully connected layer with the spherization layer. For comparison of representations in the same dimensional space, we set the dimension of the spherization layer to 1, where it is 2 in the original network. The other settings are identical to the original network. We trained both networks on the input samples for 100 epochs with 16 mini-batches, where the networks converged and achieved 100% training accuracy.

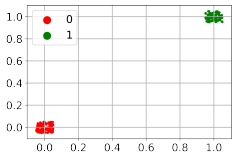


Figure 2: Input samples for the toy task

Distribution of Representations Initially, all representations are randomly distributed into two groups on a 2-dimensional space. After convergence, the feature vectors are divided into two disjointed groups, as illustrated in Figure 3. This means the spherized network locates all representations on the 2-spherical surface in both the initial and final epoch, whereas the original network spreads them out.

[2]https://github.com/weiaicunzai/pytorch-cifar100

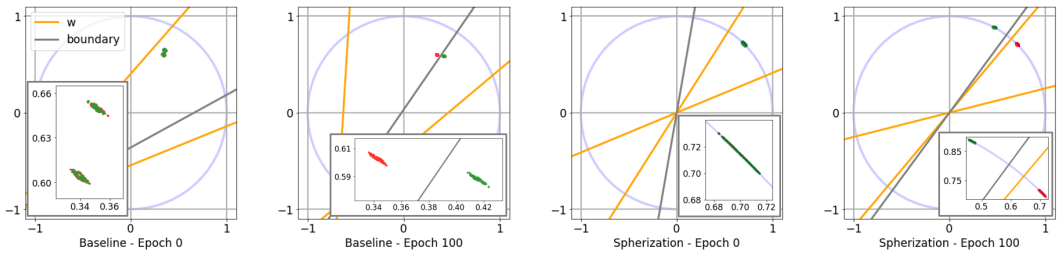


Figure 3: Visualization of Hyperplanes, Decision Boundary, and Representations in the Toy Task. (w: hyperplanes, boundary: decision boundary, red or green points: representations for label 0 or 1)

Table 1: Retention of the Training Ability on Image Classification with Various Datasets and Models (Accuarcy(%): $\mu \pm \sigma$)

Network	Dataset	Reference		Reproduced		Spherized	
		train	test	train	test	train	test
SimpleFNN [8]	MNIST	–	98.47	99.99±0.01	98.58±0.03	99.99±0.01	**98.65±0.04**
LeNet-5 [11]		–	99.05	99.55±0.09	99.10±0.05	99.79±0.09	**99.14±0.04**
VGG-11 [33]	F-MNIST	–	94.70	99.24±0.18	**94.36±0.06**	98.92±0.36	94.34±0.17
	CIFAR10	–	90.90	100.00±0.00	92.38±0.06	100.00±0.00	**92.49±0.11**
	CIFAR100	–	66.80	99.71±0.03	68.42±0.12	99.82±0.02	**69.03±0.24**

Table 2: Retention of Training Ability on Image Classification with CIFAR100 in Various Network Width and Depth Settings (Accuarcy(%): $\mu \pm \sigma$)

Depth	Width	Reference[2]		Reproduced		Spherized	
		train	test	train	test	train	test
VGG-11	16/32/64/128	–	–	79.23±5.94	60.17±0.21	77.48±5.60	**60.40±0.35**
	32/64/128/256	–	–	98.34±0.37	64.89±0.38	96.98±3.67	**65.38±0.28**
	64/128/256/512	–	–	99.71±0.03	68.42±0.12	99.82±0.02	**69.03±0.24**
	128/256/512/1024	–	–	99.90±0.00	70.53±0.35	99.93±0.00	**70.89±0.19**
	256/512/1024/1024	–	–	99.90±0.01	71.43±0.22	99.93±0.01	**71.94±0.19**
VGG-11	64/128/256/512	–	68.64	99.71±0.03	68.42±0.12	99.82±0.02	**69.03±0.24**
VGG-16		–	72.93	99.39±0.07	72.51±0.26	99.54±0.05	**72.53±0.17**
VGG-19		–	72.23	97.95±0.81	71.53±0.32	99.30±0.06	**72.17±0.33**

Hyperplanes Hyperplanes and decision boundary in the feature space are defined as follows with the parameters of the output layer:

$$\mathcal{W}_1 : \mathbf{w}_1 \cdot \mathbf{x} + \mathrm{b}_1 = 0$$
$$\mathcal{W}_2 : \mathbf{w}_2 \cdot \mathbf{x} + \mathrm{b}_2 = 0$$
$$\mathcal{D}_{12} : (\mathbf{w}_1 - \mathbf{w}_2) \cdot \mathbf{x} + (\mathrm{b}_1 - \mathrm{b}_2) = 0$$

, where $\mathcal{W}_i$ is the hyperplane determined by weight parameter $\mathbf{w}_i$ and bias b_i, and $\mathcal{D}_{ij}$ is a linear decision boundary whose points satisfy 0.5 confidence for both labels. In the spherized network, all bias terms are eliminated. In Figure 3, the hyperplanes are illustrated as yellow lines and the decision boundary as a gray line. In the spherized network, hyperplanes and decision boundary pass through the origin from the initial to the last epoch, unlike the original network. The results imply that the spherization layer can learn the correct decision boundary by changing only the angles of hyperplanes passing through the origin.

4.2 Retention of Training Ability on Image Classification Benchmarks

Experimental Settings In this task, we empirically verified that replacing an existing layer to a spherization layer still maintains the training ability of the original networks on well-known image classification tasks. We reproduced all networks and their performance on the image classification tasks with each dataset. Then, we validated our proposed method on the same settings with them, where only the last fully connected layer replaced by the spherization layer. See Appendix C for the detail about networks and datasets.

Table 3: Analysis on the Effect of Projection on Image Classification with CIFAR10 (acc.(%): accuracy, # err.: the number of errors in overlapping samples, # ovlp.: the number of overlapping samples, ratio(%): ratio of # err. to # ovlp.) (the number of test data = 10000)

Role	Operator	No-bias	Train	Test			
			acc.	acc.	# err.	# ovlp.	(ratio)
base	Original Conv.		99.47 ± 0.42	92.46 ± 0.10	0 ± 0	0 ± 0	(0.00 ± 0.00 %)
direct	Sigmoid		81.74 ± 4.48	79.03 ± 2.71	1097 ± 266	5222 ± 391	(20.75 ± 3.70 %)
	Linear		74.71 ± 1.06	72.15 ± 0.32	1631 ± 39	6849 ± 286	(23.84 ± 0.63 %)
	Cosine		77.41 ± 0.89	76.15 ± 0.86	776 ± 132	3170 ± 330	(24.39 ± 2.66 %)
indirect	SW-Softmax	✓	98.32 ± 0.07	91.51 ± 0.19	167 ± 9	7925 ± 58	(2.11 ± 0.11 %)
	LW-Softmax	✓	86.74 ± 4.06	82.12 ± 3.82	946 ± 374	7669 ± 66	(12.30 ± 4.84 %)
	CW-Softmax	✓	99.66 ± 0.04	92.29 ± 0.18	80 ± 5	7051 ± 134	(1.14 ± 0.05 %)
proposed	Spherization	✓	99.66 ± 0.05	92.38 ± 0.14	0 ± 0	499 ± 106	(0.00 ± 0.00 %)

Results and Analysis The accuracy results of original and spherized networks are compared in Table 1. The reference and reproduced results of each setting are similar. The accuracy results of the spherized networks implemented on the reproduced code are similar or slightly higher on both training and test data than those of the original networks. These results imply that the spherization layer maintains the training ability of the original network. In Table 2, the accuracy results of original and spherized networks are shown in various width and depth settings. The spherized network consistently exhibits similar or higher test accuracy over all width and depth settings. The results reveal that the spherization layer again preserves the training ability of the original network.

4.3 Analysis: Effect of Projection

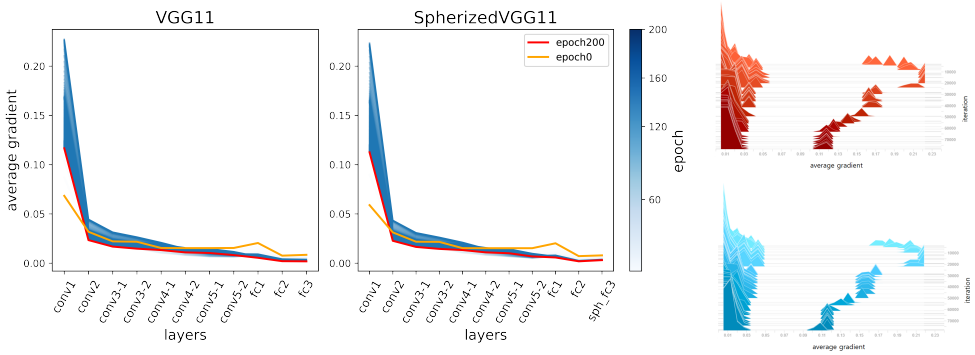
Experimental Settings We compared the method with another angle-based approach to analyze the effect of reducing information loss by projection. For comparison, we used 9-layer CNN, namely CNN-9, with the same experimental settings of [19] but reproduced in PyTorch. To apply a spherization layer, we changed the last fully connected layer in each model. We considered samples overlapped when their cosine similarity is higher than $(1 - 10^{-6})$.

Results and Analysis After training, we extracted representations from the target layer and analyzed the effect of projection. First, we count the number of representations overlapping at least one another representation. Then, we get the total errors caused by the incorrectly classified and overlapping representations. Finally, we calculate the ratio of the incorrectly classified representations to the overlapping representations. The results of them are shown in Table 3: *# ovlp.*, *# err.*, and *(ratio)*, respectively. As the result, large proportion of representations suffer the overlap problem from projection approach, and most significant errors are caused by the overlapping representations. Furthermore, representation learning relying on bias parameters (direct projection case) causes more significant errors than no-bias layer (indirection projection case) because the representations are more difficult to be distinguished than those of no-bias layer. In comparison, the spherization layer removes the errors and decreases the upperbound of errors measured by the number of overlapping representations.

4.4 Analysis: Gradient Flows

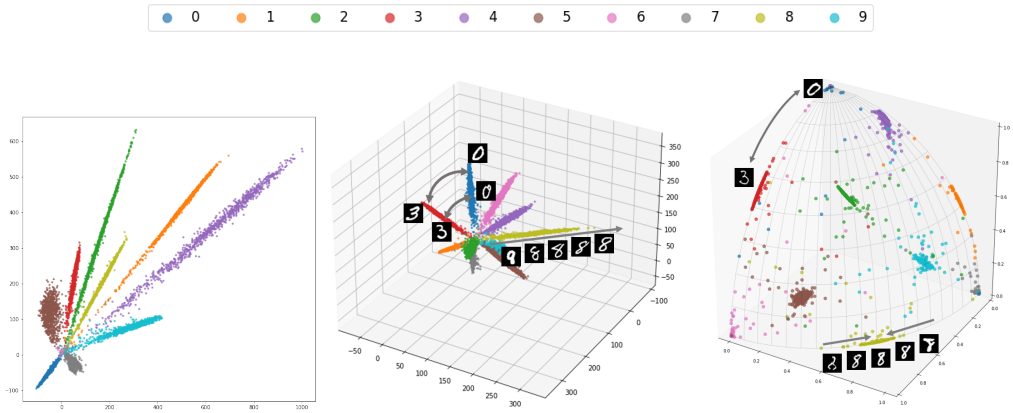
Experimental Settings The replacement of a hidden layer and the operations in the spherization layer, such as angularization function and conversion-to-Cartesian, might make the gradient flow unstable in the original networks. To empirically verify how the spherization layer affects to the gradients during training, we qualitatively compare the gradient flows of original and spherized VGG-11. They were trained on the image classification with CIFAR100. We used the same setting with Section 4.2.

Results and Analysis As shown in Figure 4, the average of absolute gradients in the original and spherized network are not very different during the whole training. Furthermore, the results show similar flows not only at the last fully connected layer (`fc3` or `sph_fc3`) but at the previous layers. This result implies the spherization layer does not destroy the gradient flows.



(a) Avg. of Absolute Gradients at Each Layer (b) Histograms

Figure 4: Analysis of Gradient Flow from Image Classification Model trained on CIFAR100. (a) The y-axis means the average of absolute gradients which occurred at each layer. The left side shows the gradient flow in VGG-11 (VGG11), and the right side shows the spherized VGG-11 (SpherizedVGG11), where the last fully connected layer is substituted with the spherization layer. (b) The histograms show the frequency of the average of absolute gradients in VGG-11 (red) and the spherized VGG-11 (cyan), respectively.



(a) Original 2D (b) Original 3D (c) Spherized 3D

Figure 5: Visualization of Feature Representations on MNIST. (a) and (b) are the visualization results of 2D and 3D feature vectors in the original networks, and (c) is the result in the spherized network

4.5 Downstream Tasks: Visualization

Experimental Settings To investigate the spherization layer locates feature vectors on the $(n+1)$-spherical surface, we implemented a simple CNN to learn 3-dimensional feature vectors for visualization. This technique is called CNN-Vis3D, which generates 3-dimensional feature vectors and shows those representations as a graph. Detailed configurations are presented in Appendix C. In the spherized network, the second fully connected layer was replaced by the spherization layer, which is located at the previous of the last fully connected layer. After training, the pre-activations before the no-bias layer were used to visualization.

Results and Analysis The results of visualization are illustrated in Figure 5. In the 2D and 3D feature visualization results of the original network, the representations are distributed over wide ranges of both scale and angles, as shown in Figure 5a and 5b. As illustrated in decoupled network [15], the angle accounts for semantic difference and the Euclidean norm accounts for intra-class variation. In the spherized network, all representations are placed on the 3-spherical surface, as shown in Figure 5c. Based on these spherized representatons and the following no-bias layer, all the roles are expressed by the angle change on the hyperspherical surface. Thus, in the spherization layer, all trained information can be used in angular similarity-based interpretation.

Table 4: Performance on the Word Analogy Test ($S_{ppl}/S_{pmi}/S_{mppl}$ ↑)

Model	SAT	U2	U4	Google	BATS	Avg.
BERT	**29.4**/28.5/**28.8**	36.0/36.0/**36.8**	**38.7**/34.7/34.3	**33.0/33.8/33.0**	32.3/**35.0**/33.2	**33.9**/33.6/**33.2**
BERT + *sph*	29.1/**29.4**/27.9	**37.3/39.0**/36.0	36.8/**35.9/35.4**	32.4/32.6/32.2	**34.0**/34.2/**33.8**	**33.9/34.2**/33.1
RoBERTa	**29.4/31.2**/29.7	35.5/**35.5/36.4**	33.6/**34.3/34.5**	32.8/33.2/30.8	30.9/31.6/30.9	32.4/33.1/32.5
RoBERTa + *sph*	29.1/29.4/**30.0**	**36.4/35.5**/34.2	**34.0/34.3**/33.3	**34.2/33.6/32.8**	**35.0/33.9/34.8**	**33.7/33.3/33.0**

Table 5: Performance of few-shot learning on Mini-ImageNet. Euclidean and Cosine mean euclidean distance and cosine similarity, respectively, which are the distance metrics that used in the experiments. (Accuarcy(%): $\mu \pm \sigma$)

Model	Test Acc.		Model	Test Acc.	
	Euclidean	Cosine		Euclidean	Cosine
ConvNet	50.29±0.18	52.87±0.18	ResNet	37.63±0.15	33.41±0.15
ConvNet + *sph*	43.41±0.16	**53.74±0.16**	ResNet + *sph*	31.77±0.13	**38.71±0.16**

4.6 Downstream Tasks: Word Analogy Test

Experimental Settings We used BERT [5] and RoBERTa [20] to conduct the word analogy test, in which angular similarity is used to predict a relation type between words. The settings were identical with [31]. We applied the spherizaton layer to the last fully connected layer of the encoder in each model. Next, we trained them on WikiText [22] for 3 epochs with 8 mini-batches, the softmax function following cross-entropy, SGD at a learning rate 0.0001 on masked-language modeling.

Results and Analysis Table 4 presents the performance evaluated with three metrics [31]. Applying the spherization layer to BERT and RoBERTa improves most average scores. The improvement implies that the spherized representations provide accurate information for angle-based distinction of word relations.

4.7 Downstream Tasks: Few-shot Learning

Experimental Settings We used ProtoNet [29] with ConvNet and ResNet for few-shot learning on Mini-ImageNet [32], in which several feature vectors are compared with the feature vector of an input image by using distance metric. This task was performed according to the guidelines in [2].

Results and Analysis Table 5 details the performance results in few-shot learning [2]. Generally, the Euclidean distance is used to discriminate feature vectors. However, the Eucliean distance also has the dispersion problem. To focus on only the angles, we trained the models with cosine similarity and compared the performances. As shown in Table 5, all spherized models with cosine similarity outperform the other models. This improvement indicates the spherized representations are useful for the angle-based metric such as cosine similarity.

5 Related Works

Semantic Analysis on the Inner Product The inner product is a crucial operator in current neural networks, in which the distance between input vector $\mathbf{x}$ and weight vector $\mathbf{w}$ is encoded. At the decoupled networks [15], the inner product is reparametrized with the norms and the angle, and the intra-class variation and the semantic difference are modeled in neural networks by decoupling them. Furthermore, the substitutes of the inner product have been proposed, where the direction of gradient or the similarity between kernels are used as the key factor instead of the inner product [14, 37]. The spherization layer can be considered to be the substitute of the inner product, which normalizes the input vectors by locating them on the hyperspherical surface. However, the spherization layer is a direct and specific method to convert feature vectors focused on the angles without information loss.

Angle-based Approach The semantic analysis on the inner product has revealed that the streams focuses on the information in the angles. The angle is a crucial factor, in which the most abundant and discriminative information is preserved [1, 12, 13, 18]. In SphereFace variants [13, 16, 17, 38],

the angular softmax that enables CNNs to learn discriminative features on angular separability was used. Furthermore, some of this angular information have been used for regularization [35, 36]. In most angle-based studies, the angular information was used indirectly by the objective function or regularization. In contrast, the spherization layer ensures the model directly learns the angular information on the hyperspherical surface.

Hyperspherical Representation Learning In some angle-based approaches, input vectors were directly projected onto the hyperphserical surface [15, 19, 21]. These hyperspherical representation learning methods normalized the input vectors to ensure models are dependent on only the angles. However, this normalization is the projection onto the hyperspherical surface, and it can be less discriminative when some points overlap after the projection. The spherization layer locates the input vectors on the hyperspherical surface without the overlap problem through the spherization.

6 Conclusion

We introduced the dispersion problem of trained information to the Euclidean norm and angle on representations. To address the dispersion problem, we proposed the *spherization layer* to learn representations by using only the angles without information loss. We used the angularization for using pre-activations as angular coordinates, conversion-to-Cartesian for locating them on the $(n+1)$-spherical surface, and no-bias training to learn representations by using only the angles. In the experiments on toy, image classification benchmarks, few-shot learning, and word analogy test, the proposed method achieved accurate learning of the decision boundary and retention of the original training ability, and improved performance in downstream tasks using angle-based information re-used or interpreted. The proposed method can be applied to numerous network layers and downstream applications. A limit of this approach is that the spherization layer should be inserted to networks in a training step to fully utilize its advantage, which restricts the use of pre-trained models trained without it. Recovering trained information on representations with sampling should be investigated in the future.

Acknowledgments and Disclosure of Funding

This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1A2C2012054), and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01842, Artificial Intelligence Graduate School Program (GIST)).

References

[1] Beidi Chen, Weiyang Liu, Zhiding Yu, Jan Kautz, Anshumali Shrivastava, Animesh Garg, and Animashree Anandkumar. Angular visual hardness. In *International Conference on Machine Learning*, pages 1637–1648. PMLR, 2020.

[2] Da Chen, Yuefeng Chen, Yuhong Li, Feng Mao, Yuan He, and Hui Xue. Self-supervised learning for few-shot image classification. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1745–1749. IEEE, 2021.

[3] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. Hyperspherical variational auto-encoders. *arXiv preprint arXiv:1804.00891*, 2018.

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

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Aiden Durrant and Georgios Leontidis. Hyperspherically regularized networks for byol improves feature uniformity and separability. *arXiv preprint arXiv:2105.00925*, 2021.

[7] Yi Hao, Nannan Wang, Jie Li, and Xinbo Gao. Hsme: Hypersphere manifold embedding for visible thermal person re-identification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 8385–8392, 2019.

[8] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Rongmei Lin, Weiyang Liu, Zhen Liu, Chen Feng, Zhiding Yu, James M Rehg, Li Xiong, and Le Song. Regularizing neural networks via minimizing hyperspherical energy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6917–6927, 2020.

[13] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. *Advances in neural information processing systems*, 31, 2018.

[14] Weiyang Liu, Zhen Liu, James M Rehg, and Le Song. Neural similarity learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Weiyang Liu, Zhen Liu, Zhiding Yu, Bo Dai, Rongmei Lin, Yisen Wang, James M Rehg, and Le Song. Decoupled networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2771–2779, 2018.

[16] Weiyang Liu, Yandong Wen, Bhiksha Raj, Rita Singh, and Adrian Weller. Sphereface revived: Unifying hyperspherical face recognition. *arXiv preprint arXiv:2109.05565*, 2021.

[17] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[18] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, volume 2, page 7, 2016.

[19] Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep hyperspherical learning. *Advances in neural information processing systems*, 30, 2017.

[20] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[21] Chunjie Luo, Jianfeng Zhan, Xiaohe Xue, Lei Wang, Rui Ren, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. In *International Conference on Artificial Neural Networks*, pages 382–391. Springer, 2018.

[22] Stephen Merity. The wikitext long term dependency language modeling dataset. *Salesforce Metamind*, 9, 2016.

[23] Pascal Mettes, Elise van der Pol, and Cees Snoek. Hyperspherical prototype networks. *Advances in neural information processing systems*, 32, 2019.

[24] Rafael S Pereira, Alexis Joly, Patrick Valduriez, and Fabio Porto. Hyperspherical embedding for novel class classification. *arXiv preprint arXiv:2102.03243*, 2021.

[25] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

[26] Jiayi Shen, Zehao Xiao, Xiantong Zhen, and Lei Zhang. Spherical zero-shot learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021.

[27] Woohyeon Shim and Minsu Cho. Circlegan: Generative adversarial learning across spherical circles. *Advances in Neural Information Processing Systems*, 33:21081–21091, 2020.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[30] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[31] Asahi Ushio, Luis Espinosa-Anke, Steven Schockaert, and Jose Camacho-Collados. Bert is to nlp what alexnet is to cv: can pre-trained language models identify analogies? *arXiv preprint arXiv:2105.04949*, 2021.

[32] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[33] Shengjie Wang, Tianyi Zhou, and Jeff Bilmes. Bias also matters: Bias attribution for deep neural network explanation. In *International Conference on Machine Learning*, pages 6659–6667. PMLR, 2019.

[34] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pages 9929–9939. PMLR, 2020.

[35] Zhennan Wang, Canqun Xiang, Wenbin Zou, and Chen Xu. Dma regularization: Enhancing discriminability of neural networks by decreasing the minimal angle. *IEEE Signal Processing Letters*, 27:2089–2093, 2020.

[36] Zhennan Wang, Canqun Xiang, Wenbin Zou, and Chen Xu. Mma regularization: Decorrelating weights of neural networks by maximizing the minimal angles. *Advances in Neural Information Processing Systems*, 33:19099–19110, 2020.

[37] Zhennan Wang, Wenbin Zou, and Chen Xu. Pr product: A substitute for inner product in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6013–6022, 2019.

[38] Yandong Wen, Weiyang Liu, Adrian Weller, Bhiksha Raj, and Rita Singh. Sphereface2: Binary classification is all you need for deep face recognition. *arXiv preprint arXiv:2108.01513*, 2021.

[39] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[40] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[41] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract and Introduction (Section 1)
   (b) Did you describe the limitations of your work? [Yes] See Conclusion (Section 6) and Appendix D
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Background (Section 2) and Spherizaton Layer (Section 3)
   (b) Did you include complete proofs of all theoretical results? [Yes] See Spherization (Section 3), Functional Correctness Test on a Toy Task in Experiments (Section 4), and Appendix A and B
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplemental material (.zip file)
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Experimental Settings in each experiments (Section 4) and Appendix C
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Tables in each experiments (Section 4)
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix and the supplemental material (.zip file)
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited them by Reference. The detail about the citation is covered at the supplemental material (.zip file)
   (b) Did you mention the license of the assets? [Yes] See the supplemental material (.zip file)
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplemental material (.zip file)
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We used open datasets. When using curated open datasets, we cited it by Reference
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used open datasets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Lower bound φ_L

To prevent the last coordinate from an extremely small value, we set a lower bound φ_L of angles. When we assume that the trigonometric value of all angular coordinates is α, $\sin\varphi = \alpha$. Based on this assumption, the lower bound φ_L can be obtained as illustrated in Eq. 12.

$$\begin{aligned} when\ \sin\varphi = \alpha,\ \alpha^n \geq \delta \ \Leftrightarrow\ \alpha = \delta^{1/n} \\ \varphi = \sin^{-1}\alpha \geq \sin^{-1}\left(\delta^{1/n}\right) \\ \therefore\ \varphi_L = \sin^{-1}\left(\delta^{1/n}\right) \end{aligned} \tag{12}$$

B Calculation Trick

To convert the angular coordinates $\boldsymbol{\varphi} \in \mathbb{R}^n$ to Cartesian coordinates $\mathbf{s} \in \mathbb{R}^{n+1}$, we first expand the dimension of $\boldsymbol{\varphi}$ by using the constant matrix $\mathbf{W}_\varphi = [\mathbf{I}_n; \mathbf{v}] \in \mathbb{R}^{n\times(n+1)}$, where $\mathbf{v} = [\mathbf{0}; 1]^\top \in \mathbb{R}^n$ and $[\cdot;\cdot]$ means concatenation. Then, we can get the expanded angular vector $\boldsymbol{\phi}$ as illustrated in Eq. 8.

When we define $\mathbf{s}'$ as illustrated in Eq. 13 from Eq. 8, we calculate the i-th coordinate s_i as shown in Eq. 14 , where $\mathbf{b}_\phi = [\mathbf{0}; -\frac{\pi}{2}]^\top \in \mathbb{R}^{n+1}$ and $\mathbf{W}_\phi \in \mathbb{R}^{(n+1)\times(n+1)}$ is an upper triangular matrix in which all diagonals are zero, $(\mathbf{W}_\phi)_{n,n+1}$ is also zero, and the other elements are ones.

$$\mathbf{s}' = \mathbf{W}_\phi^\top \ln(\sin\boldsymbol{\phi}) + \ln\left(\cos\left(\boldsymbol{\phi} + \mathbf{b}_\phi\right)\right) \tag{13}$$

$$\begin{aligned} \mathrm{s}'_{1<k<n+1} &= \sum_{i=1}^{k-1} \ln(\sin\varphi_i) + \ln(\cos\varphi_k) \\ &= \ln\left(\cos\varphi_k \prod_{i=1}^{k-1} \sin\varphi_i\right) \\ \mathrm{s}'_1 &= \ln(\cos\varphi_1) \\ \mathrm{s}'_{n+1} &= \ln\left(\prod_{i=1}^{n} \sin\varphi_i\right) \end{aligned} \tag{14}$$

Finally, we get the Cartesian coordinates $\mathbf{s}$ from this calculation trick as illustrated in Eq. 15.

$$\begin{aligned} \mathbf{s} &= r \cdot \exp(\mathbf{s}') \\ &= [r\cos\varphi_1, \cdots, r\cos\varphi_k \prod_{i=1}^{k-1} \sin\varphi_i, \cdots, r\prod_{i=1}^{n} \sin\varphi_i] \end{aligned} \tag{15}$$

C Experimental Details

Image Classification We conducted image classification experiments on MNIST [4], Fashion-MNIST [40], CIFAR10, and CIFAR100 [10]. For MNIST, we used a 3-layer neural network [8] and LeNet-5 [11]. We followed the settings in [8] and in [11], and the architectures of them are illustrated in Table 6 SimpleFNN and LeNet-5, respectively. Unlike [8], we used 256 mini-batches, Adam at a learning rate of 0.001 and weight decay of 5e-5 at the 3-layer neural network. At LeNet-5, we set 128 mini-batches, Adam at a learning rate of 0.001, and the other settings are same with [11]. For Fashion-MNIST, CIFAR10, and CIFAR100, we used VGG variants [28] with batch normalization [9] denoted as VGG-*N*. ReLU, 128 mini-batches, and SGD with momentum of 0.9 and with weight decay of 5e-4 were used as default in three datasets. The number of epochs was set to 100, 300, and 200, respectively. For Fashion-MNIST, the learning rate was initially 0.01 and it was divided by 2 at every 20 epochs. For CIFAR10, we set the initial learning rate as 0.05 and reduced it to half at every 30 epochs. For CIFAR100, the learning rate was initially 0.1 and divided by 5 at 60th, 120th, and 160th epochs. The last fully connected layer of the backbone network was replaced by the spherization layer for each task. As pre-activations used for generating angular coordinates, the range of angles was reduced when dropout. To resolve this problem, we removed the dropout in front of the spherization layer. For analysis of robustness of the spherization layer to width and depth

Table 6: Network configurations. The convolutional layer parameters are denoted as "conv(receptive field size)-(number of channels)", and the ReLU activation function is not shown for brevity [28]. The fully connected layer parameters are denoted as "FC-(number of neurons)". In CNN-Vis3D, N is set as 2 or 3 to get the feature vectors for visualization. In VGG-11 (C), $C = [C_1, C_2, C_3, C_4]$ means the number of channels at each convolutional block. N_{out} is set as the number of classes in each image classification task. The highlighted layer means where it is replaced by the spherization layer in the spherized network.

SimpleFNN	LeNet-5	CNN-Vis3D	CNN-9	VGG-11 (C)	VGG-16	VGG-19
FC-500	conv5-6	conv3-32	conv3-128	conv3-C_1	conv3-64	conv3-64
FC-300	conv5-16	conv3-32	conv3-128		conv3-64	conv3-64
FC-10	FC-120	maxpool	conv3-128	maxpool		
soft-max	FC-84	conv3-64	maxpool	conv3-C_2	conv3-128	conv3-128
	FC-10	conv3-64	conv3-192		conv3-128	conv3-128
	soft-max	maxpool	conv3-192	maxpool		
		conv3-128	conv3-192	conv3-C_3	conv3-256	conv3-256
		conv3-128	maxpool	conv3-C_3	conv3-256	conv3-256
		FC-256	conv3-256		conv3-256	conv3-256
		FC-N	conv3-256			conv3-256
		FC-10	conv3-256	maxpool		
		soft-max	maxpool	conv3-C_4	conv3-512	conv3-512
			FC-256	conv3-C_4	conv3-512	conv3-512
			FC-10		conv3-512	conv3-512
			soft-max			conv3-512
				maxpool		
				conv3-C_4	conv3-512	conv3-512
				conv3-C_4	conv3-512	conv3-512
					conv3-512	conv3-512
						conv3-512
				maxpool		
				FC-4096		
				FC-4096		
				FC-N_{out}		
				soft-max		

configurations, we used the VGG backbone again and changed the convolutional configuration of it on CIFAR100. To generate variations on width, we set the number of channels at Conv1.x, Conv2.x, Conv3.x, and Conv4.x to 16/32/64/128, 32/64/128/256, 64/128/256/512, 128/256/512/1024 and 256/512/1024/1024, respectively. The filter in Conv5.x has same number with the filter in Conv4.x (see Table 6). To give variations on depth, we used VGG-11, VGG-16, and VGG-19.

Visualization CNN-Vis3D is composed of three convolutional layer blocks whose kernel size are 32, 64, and 128, as shown in Table 6. To obtain 3-dimensional feature vectors, we set the number of neurons at the second fully connected layer, which locates before the last fully connected layer, as 3 in the original network. For a fair comparison, we replaced this layer with the spherization layer and changed the number of neurons to 2, and visualized 2-dimensional feature vectors from the original network, in which the number of neurons at the second fully connected layer is 2. In the CNN-Vis3D for 2D and 3D, named Original 2D and Original 3D respectively, batch normalization, ReLU, 64 mini-batches, cross-entropy, and Adam at a learning rate 0.001 were used as default. Unlike the original network, we set 32 mini-batches in the spherized CNN-Vis3D, called Spherized 3D. We trained original 2D, original 3D, and Spherized 3D for 20 epochs, and they converged and achieved the training accuracies(%) 98.87, 99.48, and 99.13, respectively.

D Effect of Fine-tuning

Table 7: Performance on Word Analogy Test (fine-tuning) ($S_{ppl}/S_{pmi}/S_{mppl}$ ↑)

Model	SAT	U2	U4	Google	BATS	Avg.
BERT	**29.7/32.3**/29.4	**37.3/36.0/34.2**	**39.1/34.7/34.6**	**44.8**/44.4/44.2	**44.1/41.3/40.7**	**39.0/37.7/36.6**
BERT + *sph*	29.1/29.1/**30.0**	36.8/35.1/33.3	38.2/32.6/33.3	44.6/**45.8/44.4**	40.8/38.8/39.3	37.9/36.3/36.1
RoBERTa	40.7/38.0/40.1	43.4/47.8/45.6	41.2/43.3/42.1	83.0/84.0/83.2	68.1/68.5/69.3	55.3/56.3/56.0
RoBERTa + *sph*	**41.5/42.1/42.7**	**49.6/50.4/47.8**	**46.3/46.8/46.5**	**88.0/87.0/88.4**	**71.0/70.7/70.9**	**59.3/59.4/59.3**

Table 8: Performance of Few-shot Learning on Mini-ImageNet (fine-tuning) (Accuarcy(%): $\mu \pm \sigma$)

Model	Test Acc.		Model	Test Acc.	
	Euclidean	Cosine		Euclidean	Cosine
ConvNet	**67.80±0.17**	66.60±0.17	ResNet	77.73±0.15	**78.86±0.15**
ConvNet + *sph*	51.56±0.17	61.77±0.17	ResNet + *sph*	72.18±0.15	74.57±0.15

In neural networks, the fine-tuning technique is used in natural language processing [25, 39] and computer vision [30, 41]. In this method, the pre-trained model is used and re-trained on a new target domain. Fine-tuning can achieve excellent performance. However, for spherization, a layer in the architecture should be replaced with the spherization layer, and this spherized network should be trained from scratch. If not, a mismatch between the learned representations and spherization unexpectedly affects the performance of the fine-tuned model. To verify this phenomenon, we investigated fine-tuning on the word analogy test and few-shot learning. We used the pre-trained weights from [39] for BERT and RoBERTa, and from [2] for ConvNet and ResNet. For synchronizing with spherized features, we loaded the pre-trained models and fine-tuned them on WikiText dataset [22] and Mini-ImageNet [32], respectively. See the experimental details and results on following subsections.

D.1 Word Analogy Test

Experimental Settings Unlike the few-shot learning, there are no data to re-train the pre-trained weights in word analogy test because they use language models to evaluate the performance on word analogy test. To resolve this problem, we used WikiText [22] to make fine-tuned language models. We loaded pre-trained weights BERT and RoBERTa, respectively, and trained them on WikiText for 3 epochs, 8 mini-batches, softmax function following cross-entropy, SGD at a learning rate 0.0001. For spherization, we replaced the last fully connected layer of encoder in each network with the spherization layer.

Result and Analysis The results of fine-tuning on word analogy test are illustrated in Table 7. In the original networks, the performances of both are obviously increased. The spherized BERT also have improvements but not much as the original network, while the spherized RoBERTa shows greater improvement. This result means that the angular information in the pre-trained representations might be more helpful enough to overcome the information collapse.

D.2 Few-shot Learning

Experimental Settings The experimental settings were used according to the method presented in Section 4.7. For fine-tuning, we obtained the pre-trained weights [2] for ConvNet and ResNet, respectively. In the spherized networks, the part of last fully connected layer was substituted with the spherization layer when loading these weights. Thus, the pre-trained weights of this layer were not necessary anymore.

Result and Analysis The results of fine-tuning on few-shot learning are shown in Table 8. In the original networks, the performances increase considerably. The spherized networks exhibit improvements but not as much as the original networks. This result indicates that there are some information collapse when training the spherization layer on pre-trained representations but it is not that fine-tuning is not helpful at all. To the best of our knowledge, the angular information in the pre-trained representations might be helpful to train spherized representations.