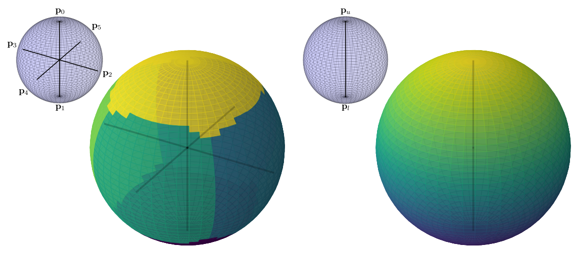
Hyperspherical Prototype Networks

Pascal Mettes [1] Elise van der Pol [2] Cees G. M. Snoek [1]

Abstract

This paper introduces hyperspherical prototype networks, which unify regression and classification by prototypes on hyperspherical output spaces. Rather than defining prototypes as the mean output vector over training examples per class, we propose hyperspheres as output spaces to define class prototypes *a priori* with large margin separation. By doing so, we do not require any prototype updating, we can handle any training size, and the output dimensionality is no longer constrained to the number of classes. Furthermore, hyperspherical prototype networks generalize to regression, by optimizing outputs as an interpolation between two prototypes on the hypersphere. Since both tasks are now defined by the same loss function, they can be jointly optimized for multi-task problems. Experimental evaluation shows the benefits of hyperspherical prototype networks for classification, regression, and their combination.



(a) Classification. (b) Regression.

Figure 1. This paper demonstrates that for (a) classification and (b) regression, output spaces do not need to be learned, they can be composed *a priori* when employing hyperspherical output spaces. This results in effective deep networks with flexible output spaces, integrated inductive biases, and the ability to optimize both tasks in the same output space without further tuning.

1. Introduction

This paper presents a class of deep networks that employ hyperspheres as output spaces with an *a priori* defined organizations. Standard classification (with softmax cross-entropy) and regression (with squared loss) are effective, but train in a fully parametric manner, ignoring known inductive biases, such as large margin separation, simplicity, and knowledge about source data (Mitchell, 1980). Moreover, they require output spaces with a fixed output size, either equal to the number of classes (classification) or a single dimension (regression). In this work, we propose networks with output spaces that can incorporate inductive biases prior to learning and have the ability to handle any output dimensionality.

Our approach is similar in spirit to recent prototype-based networks for classification, which employ a metric output space and divide this space into Voronoi cells around a prototype per class, defined as the mean location of the training examples (Guerriero et al., 2018; Hasnat et al., 2017; Jetley et al., 2015; Snell et al., 2017; Wen et al., 2016). While intuitive, this definition alters the true prototype location with each mini-batch update, which means that it requires constant re-estimation. As such, current solutions either employ coarse prototype approximations or are limited to few-shot settings. In this paper, we question this prototype definition and propose an alternative.

For classification, our notion is simple: when relying on hyperspheres as output spaces, prototypes do not need to be inferred from data. By placing prototypes as uniformly as possible on the hypershere, we obtain prototypes with large margin separation, as visualized in Fig. 1a. We outline evolutionary algorithms to position prototypes on the hypersphere prior to training. We furthermore extend the evolutionary algorithms to incorporate privileged information about classes to obtain output spaces with semantic class structures. Training and inference is in turn done through cosine similarities between examples and their class prototypes.

Prototypes that are *a priori* positioned on hyperspherical outputs extend beyond classification to regression, where we maintain two prototypes, denoting the regression bounds. The idea is to perform optimization through a relative cosine similarity of the output predictions and the two prototype bounds, as visualized in Fig. 1b. This extends standard

[1]ISIS Lab, University of Amsterdam [2]UvA-Bosch Delta Lab, University of Amsterdam. Correspondence to: Pascal Mettes <P.S.M.Mettes@uva.nl>.

regression to higher-dimensional outputs, which provides additional degrees of freedom not possible with standard regression and obtains better results. Furthermore, since we optimize both tasks with a squared cosine similarity loss, classification and regression can be performed jointly in the same output space, without the need to tune and weight the different tasks.

Experimentally, we find that hyperspherical prototype networks are effective for classification, especially when output spaces are compact and privileged information is incorporated. Furthermore, they obtain preferable results over standard approaches for regression and multi-task learning. The optimization and inference of our approach come with minimal code adjustments.

2. Model

2.1. Hyperspherical prototype classification

For classification, we are given N training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^I$ and $y_i \in C$ denote the inputs and class labels of the i^{th} training example, $C = \{1, .., K\}$ denotes the set of K class labels, and I denotes the input dimensionality. Prior to learning, the D-dimensional output space is subdivided approximately uniformly by prototypes $P = \{\mathbf{p}_1, ..., \mathbf{p}_K\}$, where each prototype $\mathbf{p}_k \in \mathbb{S}^{D-1}$ denotes a point on the hypersphere. Here, we first provide the optimization for hyperspherical prototype networks given *a priori* provided prototypes. Then we outline how to compute the hyperspherical prototypes in a data-independent manner.

Loss function and optimization. For a single training example $(\mathbf{x}_i, y_i)$, let $\mathbf{z}_i = f_\phi(\mathbf{x}_i)$ denote the D-dimensional output vector given a network $f_\phi(\cdot)$. Because we fix the organization of the output space, as opposed to learning it, we propose to train a classification network by minimizing the angle between the output vector and the prototype $\mathbf{p}_{y_i}$ for ground truth label y_i, so that the classification loss $\mathcal{L}_\text{c}$ to minimize is given as:

$$\begin{aligned}\mathcal{L}_\text{c} &= \sum_{i=1}^{N} (1 - \cos\theta_{\mathbf{z}_i, \mathbf{p}_{y_i}})^2, \\ &= \sum_{i=1}^{N} (1 - \frac{|\mathbf{z}_i \cdot \mathbf{p}_{y_i}|}{||\mathbf{z}_i||\ \ ||\mathbf{p}_{y_i}||})^2.\end{aligned} \tag{1}$$

The loss function aims to maximize the cosine similarity between the output vectors of the training examples and their corresponding class prototypes. Figure 2 provides two illustrations in a 3D output space for a training example (orange), which moves towards the hyperspherical prototype of its respective class (blue) given the cosine similarity. The higher the cosine similarity, the smaller the squared loss

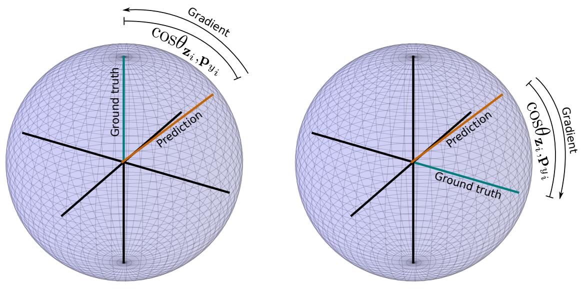

Gradient
$\cos\theta_{\mathbf{z}_i,\mathbf{p}_{y_i}}$
Ground truth
Prediction
Prediction
Ground truth
$\cos\theta_{\mathbf{z}_i,\mathbf{p}_{y_i}}$
Gradient

Figure 2. Visualization of hyperspherical prototype networks during training for classification. The main idea of our approach is for output predictions (orange) to move angularly towards the ground truth prototype (blue), using a squared cosine similarity loss between the output and class prototype ($\cos\theta_{\mathbf{z}_i, \mathbf{p}_{y_i}}$).

in the above formulation. We note that unlike common classification losses in deep networks, our loss function is only concerned with the mapping from training examples to a pre-defined layout of the output space; neither the space itself nor the prototypes within the output space need to be learned or updated.

Since the class prototypes do not require updating, our network optimization only requires a backpropagation step with respect to the training examples. Let θ_i be shorthand for $\theta_{\mathbf{z}_i, \mathbf{p}_{y_i}}$. Then the partial derivative of the loss function of Eq. 1 with respect to $\mathbf{z}_i$ is given as:

$$\frac{d}{\mathbf{z}_i}(1 - \cos\theta_i)^2 = 2 \cdot (1 - \cos\theta_i) \cdot \frac{d}{\mathbf{z}_i}(1 - \cos\theta_i), \tag{2}$$

by the chain rule, with:

$$\frac{d}{\mathbf{z}_i}(1 - \cos\theta_i) = \frac{\cos\theta_i \cdot \mathbf{z}_i}{||\mathbf{z}_i||^2} - \frac{\mathbf{p}_{y_i}}{||\mathbf{z}_i|| \cdot ||\mathbf{p}_{y_i}||}. \tag{3}$$

The remaining layers in the network are backpropagated in the conventional manner given the error backpropagation of the training examples of Eq. 2.

Hyperspherical prototype networks minimize the angle between output vectors and fixed class prototypes. As such, for a new data point $\tilde{\mathbf{x}}$, prediction is performed by computing the cosine similarity to all class prototypes and selecting the class with the highest similarity:

$$c^* = \arg\max_{c \in C} \left(\cos\theta_{f_\phi(\tilde{\mathbf{x}}), \mathbf{p}_c} \right). \tag{4}$$

Obtaining hyperspherical prototypes. The optimization hinges on the presence of class prototypes that divide the output space prior to learning. For D output dimensions and K classes, this amounts to a spherical code problem of optimally separating K classes on the D-dimensional unit-hypersphere $\mathbb{S}^{D-1}$ (Saff & Kuijlaars, 1997). For $D = 2$, this can be easily solved by splitting the unit-circle $\mathbb{S}^1$ into equal slices, separated by an angle of $\frac{2\pi}{K}$. Then, for each angle ψ, the $2D$ coordinates are obtained as $(\cos\psi, \sin\psi)$.

For $D \geq 3$, no such optimal separation algorithm exists. This is known as the Tammes problem (Tammes, 1930), for which exact solutions only exist for optimally distributing a handful of points on $\mathbb{S}^2$ and none for $\mathbb{S}^3$ and up (Musin & Tarasov, 2015). To obtain hyperspherical prototypes for any output dimension and number of classes, we first observe that the optimal set of prototypes, P^*, is the one where the largest cosine similarity between two class prototypes $\mathbf{p}_i$, $\mathbf{p}_j$ from the set is minimized:

$$P^* = \arg\min_{P' \in \mathbb{P}} \left(\max_{(k,l,k \neq l) \in C} \cos\theta_{(\mathbf{p}'_k, \mathbf{p}'_l)} \right), \qquad (5)$$

where $\mathbb{P}$ contains all sets of K vectors on $\mathbb{S}^{D-1}$. To obtain a set of prototypes on the hypersphere that adheres to Eq. 5, we outline an evolutionary algorithm for large margin separation. For a set of prototypes P, the evolutionary algorithm has a fitness function $g(P)$, which returns the minimum cosine distance between the prototype pairs in the individual. For every new generation (300 in total), we sample parents (30% from population of 3,000) using fitness proportionate selection and offsprings are produced using single-point row crossover (each parent pair produces 300 offsprings). Before insertion into the population, each new individual undergoes uniform mutation (each feature is replaced by uniform sample with $p = 0.01$). The final population size is decreased back to the original size by sampling individuals for survival, proportional to $g(P)$.

Prototypes with privileged information. The evolutionary algorithm results in a set of prototypes separated on the hypersphere in a data-independent manner. Without any prior knowledge, each class will be randomly assigned to an individual prototype. Here, we show how privileged information about classes enables class prototypes with semantic consistency. Intuitively, all classes should be far away from all other classes, but more so for dissimilar classes than similar classes. For example, the prototype angle between *cat* and *tiger* should be smaller than the angle between *cat* and *bulldozer*.

To incorporate our intuition about class semantics through privileged information (Vapnik & Izmailov, 2015) we adapt our evolutionary algorithm to exploit word2vec (Mikolov et al., 2013) representations of the class names. We note that the names of the classes generally come for free. To encourage finding hyperspherical prototypes that incorporate semantic information, we add a similarity score to the fitness function of the algorithm. This similarity score describes how similar the neighbourhoods of classes in a set

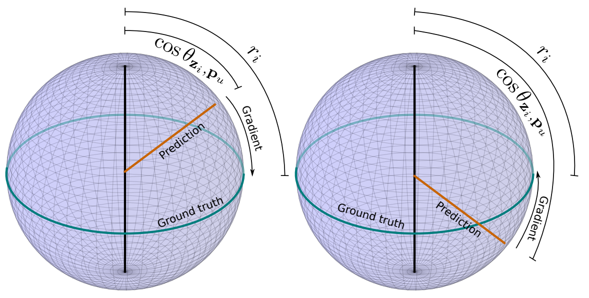

$\cos\theta_{\mathbf{z}_i,\mathbf{p}_u}$
r_i
Gradient
Prediction
Ground truth
r_i
$\cos\theta_{\mathbf{z}_i,\mathbf{p}_u}$
Ground truth
Prediction
Gradient

Figure 3. Visualization of hyperspherical prototype networks during training for regression. The main idea is for output predictions (orange) to move angularly towards the turquoise circle, which corresponds to the ground truth regression value, depending on a squared cosine loss which depends on the upper bound ($\cos\theta_{\mathbf{z}_i,\mathbf{p}_u}$) and the ground truth value (r_i).

of prototypes are to the neighbourhoods in the word2vec representations. For every class prototype $\mathbf{p}_i$, sorting based on the cosine similarity induces a ranking of 'closeness' to each prototype $\mathbf{p}_j$. We compute the distance between the word2vec order w_{ij} and the ranks induced by the prototype order p_{ij} as: $d_r(w,p) = \sum_{i,j} |\text{rank}(w_{ij}) - \text{rank}(p_{ij})|$.

The similarity score is defined to be the inverse of $d_r(w,p)$ and is added to the fitness function with a weight parameter $\lambda = 10$ that balances the importance of the semantics and the separation. Since the best ordering is given by the word2vec representations, we rely on Principal Component Analysis to reduce these to the desired number of dimensions and use the resulting representations to initialize our population. This evolutionary algorithm results in a set of prototypes on the hypersphere with large margin separation and with a semantic structure from prior knowledge.

2.2. Hyperspherical prototype regression

While current prototype-based works focus exclusively on classification, we show here that regression can be naturally handled in hyperspherical prototype networks as well. In a regression setup, we are given N training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ now denotes a real-valued regression value. The upper and lower bounds on the regression task are denoted as v_u and v_l respectively and are typically the maximum and minimum regression values of the training examples. To perform regression with hyperspherical prototypes, we first observe that training examples should no longer point towards a specific prototype as done in classification. Rather, we maintain two prototypes: $\mathbf{p}_u \in \mathbb{S}^{D-1}$ which denotes the regression upper bound and $\mathbf{p}_l \in \mathbb{S}^{D-1}$ which denotes the lower bound. Their specific direction is irrelevant, as long as the two prototypes are diametrically opposed, i.e. $\cos\theta_{\mathbf{p}_l,\mathbf{p}_u} = -1$. The idea behind hyperspherical prototype regression is to perform an interpolation between the lower and upper prototypes. We propose the following hyperspherical regression loss function:

$$\mathcal{L}_{\mathrm{r}} = \sum_{i=1}^{N} (r_i - \cos\theta_{\mathbf{z}_i,\mathbf{p}_u})^2, \tag{6}$$

$$r_i = 2 \cdot \frac{y_i - v_l}{v_u - v_l} - 1. \tag{7}$$

Eq. 6 outlines a squared loss function between two values. The first value denotes the ground truth regression value, normalized based on the upper and lower bounds. The second value denotes the cosine similarity between the output vector of the training example and the upper bound prototype. The intuition behind the loss function is shown in Fig. 3 for a 3D output space, which shows two artificial training examples with a ground truth regression value r_i is zero. Due to the symmetric nature of the cosine similarity with respect to the upper bound prototype, any output of the training example on the turquoise circle is equally correct. As such, the loss function of Eq. 6 adjusts the angle of the output prediction either away or towards the upper bound prototype, based on the difference between the expected and measured cosine similarity to the upper bound.

Our approach to regression differs from standard regression, which computes and backpropagates a loss directly on one-dimensional outputs. In the context of this work, this corresponds to an optimization on the line from $\mathbf{p}_l$ to $\mathbf{p}_u$. Our approach generalizes regression to higher dimensional output spaces. While we still aim for an interpolation between two points on the line, the ability to project to higher dimensional outputs provides additional degrees of freedom to help the regression optimization. As shown in the experiments, this generalization results in a better and more robust performance than mean squared error.

2.3. Joint regression and classification

In hyperspherical prototype networks, classification and regression are optimized in the same manner based on a squared cosine similarity loss. We show that both tasks can be optimized not only with the same base network, as is common in multi-task learning (Caruana, 1997), but can even be done in the same output space. To do so, all that is required is to place the upper and lower polar bounds for regression in opposite direction along one axis. The other axes can then be used to maximally separate the class polar prototypes for classification. Optimization is as simple as summing the losses of Eq. 1 and 6. Unlike multi-task learning on standard losses for classification and regression, our approach requires no hyperparameters to balance the tasks, as the proposed losses are inherently in the same range and have identical behaviour. This allows us to solve multiple tasks at the same time in the same space without any task-specific tuning.

3. Experimental evaluation

3.1. Classification

We start our experiments by investigating classification. Here, we evaluate the hyperspherical prototypes with large margin separation, we investigate the effect of privileged information when constructing hyperspherical prototypes, and we compare with existing prototype approaches.

Implementation details. For all our experiments, stochastic gradient descent is used as the optimizer, with a learning rate of 0.01, momentum of 0.9, weight decay of 1e-4, batch size of 128, no gradient clipping, and no pre-training. All networks are trained for 250 epochs, where after 100 and 200 epochs, the learning rate is reduced by one order of magnitude. For data augmentation, we perform random cropping and random horizontal flips.

Evaluating hyperspherical prototypes. We first evaluate the effect of hyperspherical prototypes with large margin separation using evolutionary algorithms. We perform this experiment on CIFAR-100 and ImageNet-200. CIFAR-100 consists of 60,000 images of size 32x32 from 100 classes. ImageNet-200 is a subset of ImageNet, consisting of 110,000 images of size 64x64 from 200 classes (Li et al.). For both datasets, 10,000 examples are used for testing. ImageNet-200 provides a challenging and diverse classification task, while still being condensed enough to enable broad experimentation across multiple network architectures, output dimensions, and hyperspherical prototypes. We compare to two baselines. The first consists of one-hot vectors on the C-dimensional simplex for C classes, as proposed in (Chintala et al., 2017). This baseline adheres to the class encoding in softmax cross-entropy optimization on the simplex. The second baseline consists of word2vec vectors for each class based on their name (Mikolov et al., 2013). This experiment is performed for three output dimensionalities $\{10, 25, 100\}$.

The results with a ResNet-32 network (He et al., 2016) are shown in Table 1. For both CIFAR-100 and ImageNet-200, the hyperspherical prototypes obtain the highest scores when the output size is equal to the number of classes. The baseline with standard one-hot vectors on the simplex can not handle fewer output dimensions. Our approach can, and maintains most of the accuracy when using only a quarter of the output space. For CIFAR-100, the hyperspherical prototypes perform ten to over fifteen percent points better than the baseline with word2vec prototypes. On ImageNet-200, the behavior is similar. When using even fewer output dimensions, the relative accuracy of our approach increases further. These results show that hyperspherical prototype networks can handle any output dimensionality and outperform prototype alternatives.

	CIFAR-100			ImageNet-200		
Dimensions	10	25	100	25	50	200
One-hot	-	-	60.9	-	-	32.9
Word2vec	29.0	45.4	56.1	21.2	27.7	30.0
This paper	**49.5**	**60.9**	**64.1**	**36.6**	**42.8**	**44.9**

Table 1. Accuracy (%) of our hyperspherical prototypes compared to baseline prototypes within the same framework on CIFAR-100 and ImageNet-200, using ResNet-32 as architecture. Hyperspherical prototypes with large margin separation can handle any output dimensionality, unlike standard one-hot class encodings, while obtaining the best scores across dimensionality and dataset.

	CIFAR-100			ImageNet-200		
Dimensions	10	25	100	25	50	200
One-hot	-	-	72.2	-	-	**60.4**
Word2vec	36.6	61.0	71.9	35.9	46.0	56.6
This paper	**60.9**	**67.1**	**72.3**	**55.6**	**56.3**	59.3

Table 2. Accuracy (%) of our hyperspherical prototypes compared to baseline prototypes within the same framework on CIFAR-100 and ImageNet-200, using DenseNet-121 as architecture. Akin to the results for ResNet-32, hyperspherical prototypes yield a favorable accuracy across dimensionality and dataset, highlighting their effectiveness.

We also investigate the effectiveness of our approach with a deeper and more recent DenseNet-121 architecture (Huang et al., 2017). Different to the experiments on ResNet-32, we observe that the baseline prototypes now obtain results similar to our approach when the number of output dimensions equal the number of classes, which can be explained by the significant increase in model depth and connectivity. Interestingly, we observe that when using smaller output spaces, our approach retains even more of the classification performance relatively and absolutely. This does not hold for the word2vec prototypes, as is also the case with ResNet-32. Overall, the experiments show that hyperspherical prototype networks are effective across multiple datasets and network architectures, especially when using compact output spaces.

Prototypes with privileged information. Second, we evaluate the effect of incorporating privileged information from class names when obtaining the hyperspherical prototypes. With privileged information, a semantic class structure can be enforced in the output space. We envision that such a semantic class structure makes optimization easier and is beneficial when learning from fewer output dimensions. To that end, we investigate classification on CIFAR-100 using restricted output dimensions. We rely on a ResNet-32 architecture with a wide range of output dimensions.

	CIFAR-100					
Dimensions	3	5	10	25	50	100
This paper	4.9	24.4	49.5	60.9	63.0	64.1
+ Privileged info	**8.8**	**30.3**	**53.4**	**61.5**	**64.2**	**64.4**

Table 3. Accuracy (%) of hyperspherical prototype networks without and with semantic class structures from privileged information on CIFAR-100 using ResNet-32 as network architecture. Incorporating class semantics into hyperspherical prototypes results in better classification accuracy, especially when output spaces are low-dimensional.

In Table 3, we provide the results of our approach both without and with privileged information on CIFAR-100. The further the output space is reduced, the better the relative accuracy of the hyperspherical prototypes with semantic class structure. When using only five or ten output dimensions, incorporating class semantics through privileged information improves results by four to six percent points. Across all output dimensionalities, we observe a preference for a semantic class structure, highlighting its effectiveness in hyperspherical prototype networks.

Comparison to other prototype networks. Third, we compare to the standard in prototype-based networks, where prototypes are defined as the class means and the Euclidean distance is used (Guerriero et al., 2018; Jetley et al., 2015; Snell et al., 2017; Wen et al., 2016). We have opted to compare to the work of Guerriero et al. (2018), since it can handle any number of training examples and any output dimensionality, akin to our approach. For the classification comparison we follow (Guerriero et al., 2018) and report on CIFAR-100. We have run the baseline graciously provided by the authors with the same hyperparameter settings and network architecture as used in our paper, be it that we report all their settings for prototype computations: mean condensation, mean decay, and online mean updates.

In Fig. 4, we provide the test accuracy as a function of the training epochs on CIFAR-100. Overall, our approach provides multiple benefits over (Guerriero et al., 2018):

1. The convergence of hyperspherical prototype networks is faster and reaches better results than the baselines. Fig. 4 shows that our approach obtains a higher test accuracy than the baselines after few training epochs. During the first 100 epochs, our approach performs between 5 and 10 percent points better than the baseline and the final improvement is 2 to 4 percent points.

2. The test accuracy of hyperspherical prototype networks is smoother than the baseline. Fig. 4 shows how our approach obtains a test accuracy that gradually improves over the training epochs and clearly converges, while

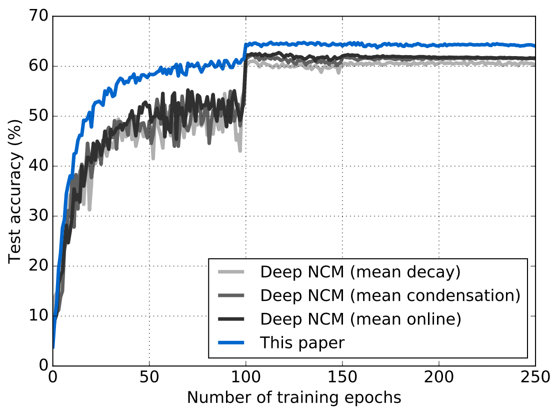


Figure 4. Classification comparison of hyperspherical prototype networks to (Guerriero et al., 2018). Our approach outperforms the baseline across all their settings when using the same hyperparameter settings and network architecture, highlighting its effectiveness, while our approach alleviates the need to compute and update the class prototypes themselves.

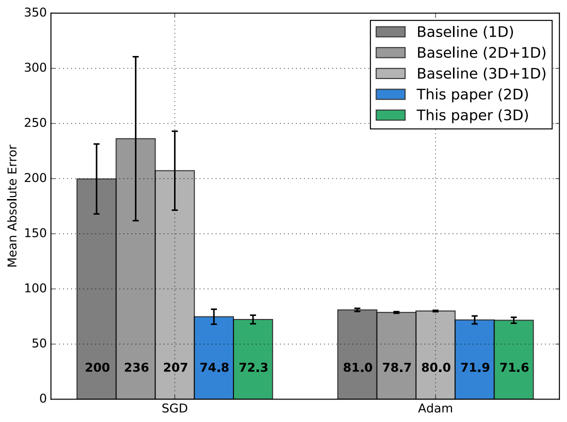


Figure 5. Mean absolute error (in years) for predicting the creation year of paintings from the OmniArt dataset (Strezoski & Worring, 2017). Our approach outperforms standard squared loss regression with statistical significance using both SGD and Adam as optimizers.

the test accuracy of the baseline behaves more erratic between training epochs.

3. The optimization of hyperspherical prototype networks is computationally easier and more efficient. After a feed forward step through the network, each training example only needs to compute the cosine similarity with respect to their respective class prototypes. The baseline needs to compute a distance to *all* classes, followed by a softmax. Furthermore, the class prototypes require constant updating, while our prototypes remain fixed. Lastly, hyperspherical prototype networks are easier to implement and require only a few lines of code given pre-computed prototypes, which does not hold for other prototype-based networks.

Overall, we conclude that hyperspherical prototype networks provide a fast, effective, and easy approach to prototype-based deep networks.

3.2. Regression

Next, we evaluate hyperspherical prototype networks for regression. We evaluate on the challenging task of predicting the creation year of paintings. We focus on paintings from the 20^{th} century available as part of the OmniArt dataset (Strezoski & Worring, 2017). This results in a dataset of 15,000 training examples and 8,353 test examples. We employ a ResNet-16 architecture (He et al., 2016) trained akin to the classification setup. The Mean Absolute Error is adopted for evaluation. We compare to a squared loss regression baseline, where we normalize and clamp the outputs between 0 and 1 using the upper and lower bounds to provide a fair comparison. For the baseline, we also include variants where the output layer has more dimensions, with an additional layer to a one-dimensional output.

Figure 5 provides an overview of the regression accuracy of our approach compared to the baseline. When using Stochastic Gradient Descent as the optimizer, the squared loss regression baselines all fail to converge, resulting in high error rates. Our approach using SGD does converge and obtains improved results. Given the large difference in accuracy, we also reran all experiments using Adam (Kingma & Ba, 2014). With this setting, the baselines perform better. However, our approach also improves in regression accuracy. We find that, using Adam, our approach with a two-dimensional output significantly outperforms the baseline with a two-dimensional plus an additional one-dimensional layer ($p = 0.02$ for a two sample t-test, H_0 = both samples have same mean). This also holds for the comparison with three dimensions ($p = 0.01$). With our approach, we observe that using three dimensions over two results in slightly better accuracy, but not significantly ($p = 0.9$). We conclude that hyperspherical prototype networks provide an effective and robust solution for regression.

3.3. Joint classification and regression

Finally, we investigate the possibility and potential of optimizing classification and regression tasks jointly in the same output space. Since both tasks are modeled as squared cosine similarity losses, an output space can be structured for both tasks by assigning different tasks to different subspaces of the hypersphere. We first investigate

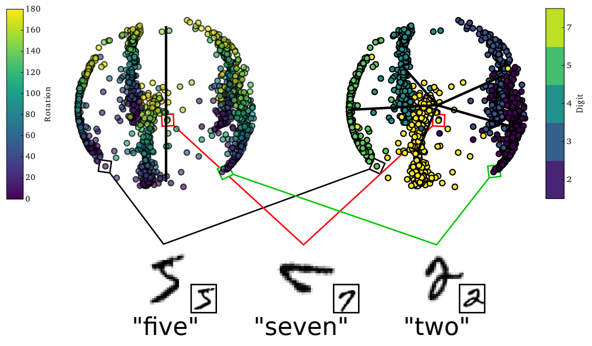


Figure 6. Joint regression and classification in the same output space on a rotated MNIST subset as a proof of concept. The same output space is shown twice for ease of visualization. On the left, the examples are linearly interpolated along the z-axis. On the right, the examples are grouped based on their class assignment on the (x, y)-plane. Since both regression and classification are modeled as a squared cosine similarity loss, the output space can be disentangled into multiple tasks and visualized accordingly.

a simple problem for which we can visualize the output space with only three dimensions. For this, we utilize MNIST, focusing jointly on class prediction and image rotation estimation. After that, we focus our attention again on the OmniArt dataset (Strezoski & Worring, 2017), where we evaluate the challenging task of jointly predicting creation year (regression) and art style (classification) on high-dimensional hyperspheres.

Rotated MNIST. For the first proof of concept, we use MNIST, where we aim to both classify the digits and regress on the rotation of the examples. We use the digits 2, 3, 4, 5, and 7 and apply a random rotation between 0 and 180 degrees to each example. The other digits were not of interest given the rotational range. We employ a 3-dimensional output space, where the classes are optimally separated along the (x, y)-plane and the regression bounds are projected along the z-axis. A simple network is used with two convolutional and two fully connected layers.

Fig. 6 shows the same output space twice, once color coded for regression (left) once for classification (right). The figure shows how in one output space, both image rotations and image classes can be modeled. Along the z-axis, images are gradually rotated, while the (x, y)-plane is split into maximally separated slices representing the image classes. This proof of concept clearly shows that both tasks can be modeled jointly in the same output space and we will use this outcome to investigate joint classification and regression in higher-dimensional output spaces.

Predicting creation year and art style. For the final joint regression and classification experiment, we aim to regress on the creation year on the OmniArt dataset, akin to our regression-only experiment, as well as classify the art style. There are in total 46 art style categories, which denote the school to which the artwork belongs. Example styles in the dataset include the *Dutch* and *French* art schools.

SGD	Creation year (MAE ↓)	Art style (Accuracy ↑)
Multi-task baseline	N/A	40.4
This paper	**71.1**	**51.6**
Adam	Creation year (MAE ↓)	Art style (Accuracy ↑)
Multi-task baseline	336.2	48.8
This paper	**75.3**	**51.3**

Table 4. Regression error (MAE) and classification accuracy (%) for joint creation year and art style prediction on OmniArt. For both optimizers, our approach is preferred for the multi-task optimization, since no tuning between the tasks is required.

We have trained for joint creation year and art style prediction using a ResNet-16 architecture, akin to the regression-only experiment. We experiment using both SGD and Adam as optimizers, since the baseline regression has shown to be unstable for SGD. We compare to a standard multi-task baseline, which uses the same network and hyperparameter settings, but with squared loss for regression and softmax cross-entropy for classification.

The results are shown in Table 4 for both creation year (mean absolute error) and art style (classification accuracy). When using SGD as the optimizer, our approach learns to optimize for both tasks and improves over an independent optimization in the same output space for both tasks (47.3 accuracy for style prediction alone). The baseline however fails to yield any regression scores due to exploding gradients, while art style yields subpar results. For Adam, we again observe that our approach is preferred over the baseline. Noteworthy is the high error for creation year for the baseline. Upon closer inspection, we found that the loss for the baseline multi-task approach was dominated by the classification loss. As such, the regression loss contributed only marginally, resulting in high regression errors during inference, hence the need for additional tuning in the baseline setting. In hyperspherical prototype networks, both tasks are modelled with the same squared cosine similarity loss, which means that a tuning of different tasks is not required in a multi-task setting.

4. Related work

Our approach relates to prototype-based networks, which have recently gained traction under various names, including proxies (Movshovitz-Attias et al., 2017), means (Guerriero et al., 2018), prototypical concepts (Jetley et al., 2015), and prototypes (Snell et al., 2017). In general, these works adhere to the Nearest Mean Classifier paradigm (Mensink et al., 2013) by assigning training examples to a vector in the output space of the network, which is defined to be the mean vector of the training examples. A few works have also investigated multiple prototypes per class (Movshovitz-Attias et al., 2017; Yang et al., 2018). Prototype-based networks result in a simple output layout (Wen et al., 2016) and generalize quickly to new classes (Guerriero et al., 2018; Snell et al., 2017; Yang et al., 2018).

While promising, the training of prototype networks is currently faced with a chicken-or-egg dilemma. Training examples are mapped to class prototypes, while class prototypes are defined as the mean of the training examples. Because the projection from input to output changes continuously during network training, the true location of the prototypes changes with each mini-batch update. Obtaining the true location of the prototypes is expensive, as it requires a pass over the complete dataset. As such, prototype networks currently either focus on the few-shot regime (Boney & Ilin, 2017; Snell et al., 2017), or on approximating the prototypes, e.g. by alternating the example mapping and prototype learning (Hasnat et al., 2017) or by updating the prototypes online as a function of the mini-batches (Guerriero et al., 2018). We propose to bypass the prototype learning altogether by structuring the output space prior to training. By defining prototypes as points on the hypersphere, we are able to separate them with large margins *a priori* through evolutionary algorithms. The network optimization simplifies to minimizing a cosine distance between training examples and their corresponding prototype, alleviating the need to continuously obtain and learn prototypes. We also note that we generalize beyond classification to regression using the same optimization and loss function.

The work of Perrot and Habard (Perrot & Habrard, 2015) relates to our approach by employing pre-defined prototypes in Euclidean space in the context of metric learning, while we employ prototypes with large margin separation on the unit hypersphere for classification and regression in deep networks. Bojanowski and Joulin (Bojanowski & Joulin, 2017) showed that unsupervised learning is possible by projecting examples to random prototypes on the unit hypersphere and updating prototype assignments. Here, we similarly investigate hyperspherical prototypes, but do so in a supervised setting, without the need to perform any prototype updating. Recent work of Liu et al. (Liu et al., 2018) similarly aims for large margin polar separation of class vectors in the output space by adding the separation as a regularization to a softmax-based deep network. Here, we take this notion further by fixing highly separated prototypes prior to learning, rather than steering them during training, while enabling the use of prototypes in regression.

Several related works have previously investigated the merit of optimizing based on angles over distances in deep networks. Liu et al. (2016), for example, aim to improve the separation in softmax cross-entropy by increasing the angular margin between classes. In similar fashion, several works project network outputs to the hypersphere for classification through ℓ_2 normalization, which forces softmax cross-entropy to optimize for angular separation (Hasnat et al., 2017; Liu et al., 2017a; Wang et al., 2018; Zheng et al., 2018). The work of (Gidaris & Komodakis, 2018) shows that using cosine similarity in the output helps generalization to new classes. The potential of angular similarities has also been investigated in other layers of deep networks (Liu et al., 2017b; Luo et al., 2017). In this work, we also focus on angular separation in deep networks, but do so from another perspective, namely in a prototype-based setting.

5. Conclusions

This paper proposes hyperspherical prototype networks for classification and regression. The key insight is that class prototypes should not be a function of the training examples, as is currently the standard, because it creates a chicken-or-egg dilemma during training. Indeed, when network weights are altered for training examples to move towards class prototypes in the output space, the class prototype locations alter too. We propose to treat the output space as a hypersphere instead, which enables us to distribute prototypes with large margin separation without the need for any training data and specification prior to learning. Due to the general nature of hyperspherical prototype networks, we introduce extensions to deal with privileged information about class semantics, continuous output values, and joint task optimization in one and the same output space. Empirically, we have learned that hyperspherical prototypes are effective, fast to train, and easy to implement, resulting in flexible deep networks that can handle regression and classification tasks in compact output spaces.

Acknowledgements

This work is supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number D17PC00343. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing endorsements, either expressed or Thanks to Gjorgji Strezoski and Thomas Mensink for help with dataset preparation and experimentation.

References

Bojanowski, P. and Joulin, A. Unsupervised learning by predicting noise. *ICML*, 2017.

Boney, R. and Ilin, A. Semi-supervised few-shot learning with prototypical networks. *CoRR*, 2017.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chintala, S., Szlam, A., Tian, Y., Tygert, M., Zaremba, W., et al. Scale-invariant learning and convolutional networks. *Applied and Computational Harmonic Analysis*, 42(1): 154–166, 2017.

Gidaris, S. and Komodakis, N. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018.

Guerriero, S., Caputo, B., and Mensink, T. Deep nearest class mean classifiers. In *ICLR, Worskhop Track*, 2018.

Hasnat, M., Bohné, J., Milgram, J., Gentric, S., and Chen, L. von mises-fisher mixture model-based deep learning: Application to face verification. *CoRR*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Jetley, S., Romera-Paredes, B., Jayasumana, S., and Torr, P. Prototypical priors: From improving classification to zero-shot learning. *BMVC*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2014.

Li, F.-F., Karpathy, A., and Johnson, J. `https://tiny-imagenet.herokuapp.com`.

Liu, W., Wen, Y., Yu, Z., and Yang, M. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016.

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017a.

Liu, W., Zhang, Y.-M., Li, X., Yu, Z., Dai, B., Zhao, T., and Song, L. Deep hyperspherical learning. In *NeurIPS*, 2017b.

Liu, W., Lin, R., Liu, Z., Liu, L., Yu, Z., Dai, B., and Song, L. Learning towards minimum hyperspherical energy. *NeurIPS*, 2018.

Luo, C., Zhan, J., Wang, L., and Yang, Q. Cosine normalization: Using cosine similarity instead of dot product in neural networks. *CoRR*, 2017.

Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. Distance-based image classification: Generalizing to new classes at near-zero cost. *TPAMI*, 35(11):2624–2637, 2013.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, 2013.

Mitchell, T. M. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

Movshovitz-Attias, Y., Toshev, A., Leung, T. K., Ioffe, S., and Singh, S. No fuss distance metric learning using proxies. *CVPR*, 2017.

Musin, O. R. and Tarasov, A. S. The tammes problem for n=14. *Experimental Mathematics*, 24(4):460–468, 2015.

Perrot, M. and Habrard, A. Regressive virtual metric learning. In *NeurIPS*, 2015.

Saff, E. and Kuijlaars, A. Distributing many points on a sphere. *The mathematical intelligencer*, 19(1), 1997.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

Strezoski, G. and Worring, M. Omniart: multi-task deep learning for artistic data analysis. *CoRR*, 2017.

Tammes, P. On the origin of number and arrangement of the places of exit on the surface of pollen-grains. *Recueil des travaux botaniques néerlandais*, 27(1):1–84, 1930.

Vapnik, V. and Izmailov, R. Learning using privileged information: similarity control and knowledge transfer. *JMLR*, 16(2023-2049):2, 2015.

Wang, H., Wang, Y., Zhou, Z., Ji, X., Li, Z., Gong, D., Zhou, J., and Liu, W. Cosface: Large margin cosine loss for deep face recognition. *CVPR*, 2018.

Wen, Y., Zhang, K., Li, Z., and Qiao, Y. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016.

Yang, H.-M., Zhang, X.-Y., Yin, F., and Liu, C.-L. Robust classification with convolutional prototype learning. In *CVPR*, 2018.

Zheng, Y., Pal, D. K., and Savvides, M. Ring loss: Convex feature normalization for face recognition. In *CVPR*, 2018.