Weak Metric Learning

Brian Chen
Department of Computer Science
Columbia University
New York, NY 10027
bc2924@columbia.edu

Nakul Verma
Department of Computer Science
Columbia University
New York, NY 10027
verma@cs.columbia.edu

Abstract

Current approaches to distance metric learning are unable to retain the low-level structure of data; for instance, contrastive loss approaches are able to separate between broadly similar and dissimilar classes, but are unable to retain the sub-structure within a cluster of similar points in the resultant embedding. We explore two approaches to resolving this issue: one based on a joint learning approach that directly optimizes for substructure preservation, and another that augments the classic contrastive loss function with a t-SNE inspired penalty term. We show empiric improvements in the preservation of substructures under both methods. Such an algorithm has immediate applications in exploratory data analysis, e.g. in biomedical or neuroscience contexts, where our procedure can improve upon an initial coarse visualization and reveal deeper structures that were not immediately known[1].

1 Introduction

Consider a typical sequence in exploratory data analysis: a topic expert (e.g. a neuroscientist) attempting to understand regions of brain activity during different types of mouse activity. A neuroscientist might be given some high-dimensional data representing brain activity, and wishes to understand the geometric structure of this data. An initial visualization may be done (e.g. with an algorithm like t-SNE), which may reveal some coarse groupings or modalities present within the data. However, when the raw measurement space is poor, our visualization may not reflect anything meaningful about the data. A topic expert however may have some background knowledge (e.g. some prior understanding of the salient regions of the brain), which the expert wishes to use to improve the visualization of the data. Exploratory data analysis should ideally reveal *more* about the data, rather than reflect what we already know. As such, the task is to (1) be able to incorporate background knowledge about a dataset within our visualization of our data, while also (2) maintaining any interesting substructure from the original measurement space.

Contrastive loss functions are well-suited for this type of task. Given a set of similar and dissimilar pairs, contrastive loss penalizes similar pairs if they are embedded far apart, and penalizes dissimilar pairs if they are too close together. However, standard contrastive loss-based metric learning struggles to retain structural separation within clusters.

The aim of this project is to develop a metric learning algorithm that not only maintains separation between clusters of points with distinct labels but also preserves finer distinctions within each cluster. We investigated two main approaches to this task: (1) a combined classification approach that aims to increase the size of the output layer even if the number of different labels provided is low, and

[1] Code for this project can be found at https://github.com/brianxchen/weak-metric-learning/tree/main

(2) a separate metric learning phase that aims to augment the contrastive loss penalty with a term encouraging substructure maintenance.

2 Prior Work

Other strategies for parametric structure discovery are either unable to incorporate pairwise supervision [4], or require the number of clusters to be known in advance [8].

Our method is fundamentally different in that it directly attempts to optimize for substructure. Pairwise supervision is costly and may not be fully representative of the initial data, which approach (2) aims to improve upon. The combined version in approach (1) is somewhat more agnostic to the number of clusters given, and can hence be a superior alternative to hard-coding the number of clusters (as in methods like DEC).

3 A joint learning approach

In this section we propose a simple augmentation that can be applied to any loss function that can maintain the separation between coarsely-labeled classes while also separating the subclasses. This is a joint learning approach that directly optimizes for a target number of classes.

The motivation is as such: consider a dataset that has 10 true classes, but we happen to only have a coarse level labeling of 5 broad classes. In this case, if we train a classifier on these 5 labels, and then extract the embedding layer of the data from the trained output, it is well-known that the data should form orthogonal 'clusters' corresponding to each of the five learned classes. If we then run t-SNE on these embeddings, we can see a clear visual separation between these five broad classes.

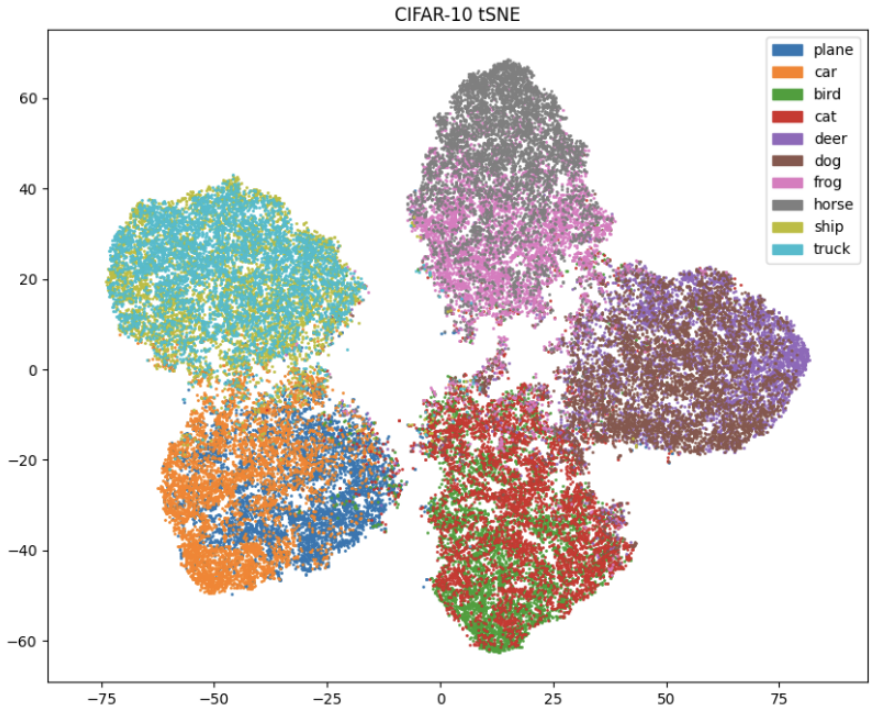

Figure 1: If a classifier is learned on five labels of a truly 10-label class, each of the combined classes will get mixed together in five distinct clusters. Here, each of the five large blobs of points corresponds to points that have been assigned the same (coarse) label, e.g. 'plane' and 'car' labels have been merged. The points have been colored with the original labels to highlight the mixture of classes, but the classifier was trained only on the merged labels.

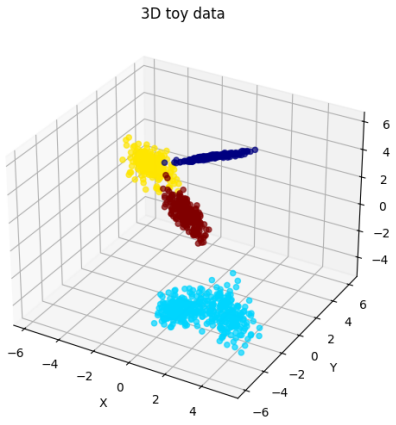

Figure 2: A toy dataset in $\mathbb{R}^3$ where two classes at the bottom have been 'merged'.

We are interested in the question: is there a way to augment the loss function in a way that encourages the sub-classes in each of these combined clusters to remain separate, even if we are only given a coarse set of labels?

3.1 A class uniformity-encouraging loss function

If we operate under the assumption that a dataset with 5 distinct labels truly does have, say, 10 "fine" classes, a natural assumption to make is that the data is uniformly distributed between these 10 "fine" classes. Under this assumption then we define a loss function that calculates the KL divergence between the batchwise distribution of labels predicted from the classifier and a uniform distribution between C classes.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{base}} + \lambda \cdot \mathcal{L}_{\text{uniform}} \tag{1}$$

$$\mathcal{L}_{\text{uniform}} = \sum_{i=1}^{C} \bar{p}_i \left(\log \bar{p}_i - \log \frac{1}{C} \right) \tag{2}$$

where $\overline{p}_i$ is the averaged activation of the softmax layer of the classifier,

$$\bar{p}_i = \frac{1}{N} \sum_{n=1}^{N} \text{softmax}(\mathbf{f}(x_n))_i$$

$\mathcal{L}_{\text{base}}$ can be any typical classification loss, e.g. cross entropy. λ is a hyperparameter controlling the weight of the uniformity term. C is the number of "true" classes that we estimate that the data set actually has; e.g. we can encourage a dataset with 5 coarse labels to learn 10 "fine" labels (we discuss the choice of this parameter in Section 3.4; in general the performance of the algorithm is somewhat agnostic to the choice of C, provided that C is large enough). N is the batch size and should be relatively large to get a good estimate of the distribution of labels. Finally, $\mathbf{f}(x_n)$ are the output logits for sample n.

3.2 A proof of concept of the uniformity penalty

To show the behavior of a classifier with and without the uniformity penalty, consider the following dataset in $\mathbb{R}^3$.

The light blue cluster at the bottom of the plot is comprised of two separate clusters generated by two separate Gaussian distributions, but their labels have been combined. This represents a situation in which the coarse labels given (4 classes) may mask the existence of an additional 5th class that we just did not have the resolution to see. If we learn a classifier on the original 4 classes, and then extract the embedding layer of this classifier, then the two clusters at the bottom get merged.

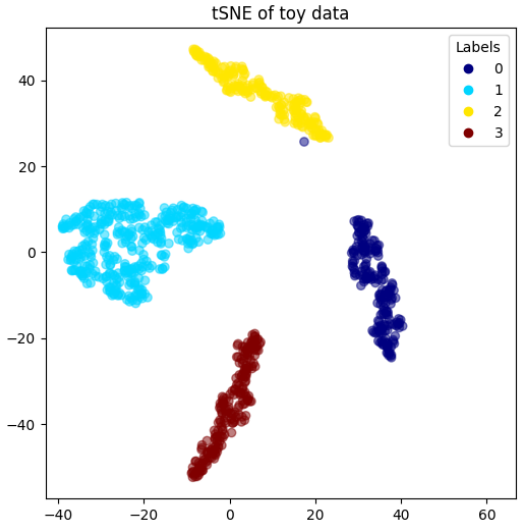

Figure 3: a t-SNE plot of the embedding layer of a neural network learned on just the original 4 classes. The two clusters comprising the light-blue points have been merged and any separation between the two in the original space has been lost.

However, if we augment our cross-entropy loss function with our uniformity penalty, then the resultant embedding preserves the separation between the two sub-clusters within the light blue cluster.

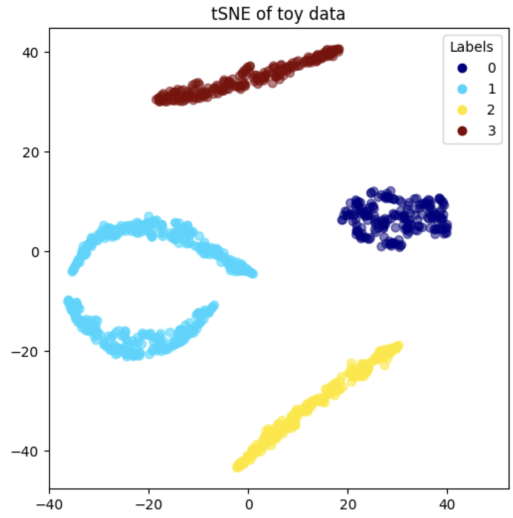

Figure 4: a t-SNE plot of the embedding layer of a neural network learned on the 4 original labels, but told to assume that a larger number of classes exist. The extra class is "learned" via the uniformity penalty and has maintained separation in the learned embedding.

This gives immediate indication that such a uniformity penalty may be effective in "learning" additional classes that exist in the original input space but may have been lost during the classification step.

3.3 Evaluation

We use neighborhood purity as a measure of the quality of the resultant separation. We define

$$\texttt{Purity}(n) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{k}\sum_{j\in\mathcal{N}_k(i)}\mathbb{I}(y_j = y_i)$$

where n is number of data points, $\mathcal{N}_k(i)$ are the k nearest neighbors of i, and y_i is the label of the i-th point. If a dataset of x unique true labels has been labeled with y labels, with $y < x$, we expect a neighborhood purity of approximately y/x since there should be no distinction between the combined classes, so the neighborhood purity should be based essentially on random choice between the combined clusters. We aim to improve upon this y/x lower bound.

Returning to our CIFAR-10 example, given a dataset where the original 10 classes have had their labels combined into 5 merged classes, we demonstrate the improvements in separation between traditional classification and our augmented loss function.

TODO: this figure is really hard to read, fix

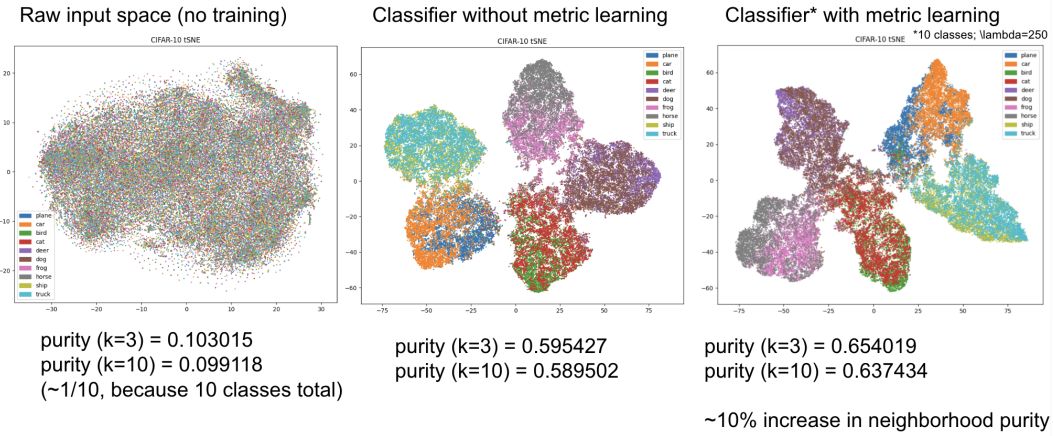

Figure 5: A comparison of CIFAR-10 embedding separation without any classification, with just the classification but no uniformity penalty, and the classification with the uniformity penalty augmentation. The uniformity penalty is able to improve the neighborhood purity of the resultant embeddings by about 10%.

3.4 The effect of the choice of C on output quality

In an unsupervised or weakly supervised setting, we may not know the true number of classes C. For instance, imagine that one is given the Fashion-MNIST dataset, labeled by "shoes", "tops", and "pants", but not knowing exactly what sub-classes (t-shirts, dresses, boots, etc.) there are. In this case, one may need to simply guess the number of true classes – e.g. we may guess that there are 20 different types of clothing represented within the Fashion-MNIST dataset.

From initial observations, the uniformity penalty is somewhat agnostic to the choice of C, allowing for flexibility in guessing the true number of clusters.

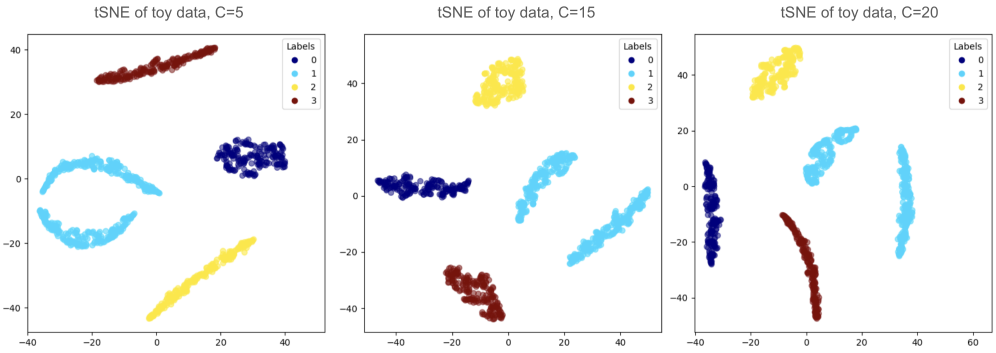

Figure 6: The quality of the output is somewhat agnostic to the choice of the number of clusters C that we choose in our uniformity penalty.

3.5 Alternative approaches

We experimented with a number of other loss functions intended to emphasize the discovery of more classes within a coarsely-labeled dataset. These were empirically less effective based on preliminary experiments, but the ideas may be useful for future analysis. The following is a laundry list of augmentations to the loss function that had potential but did not perform as well as the KL-divergence based approach.

One approach was to directly threshold and penalize if any one neuron was activated too much, e.g. more than 20% activation of a neuron should trigger some penalty. Here, n_c is the total number of predictions for class c, C is a user-specified total number of classes, and τ is some overuse threshold (e.g. 20%).

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \lambda \sum_{c=1}^{C} \max\left(\frac{n_c}{\sum_{c'} n_{c'}} - \tau, 0\right)$$

We found that this method was inconsistent in learning a separation between sub-classes like we were looking for. It was also unclear how the choice of the threshold penalty should be made.

Another approach that we tried was a ridge-regression inspired penalty – adding a L^2 penalty onto the activation layer in an attempt to prevent any one neuron from getting over-activated over a batch. This loss function took the form

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \frac{1}{N} \sum_{j=1}^{N} f_i(x_j)^2$$

where $f_i(x_j)$ is the activation of the i-th neuron on the j-th point in the batch. This approach was initially appealing due to its lack of external hyperparameters to tune for (unlike the uniformity approach, which relies on assigning C in advance); however, the resultant embeddings were not consistently separated.

4 A contrastive loss approach

4.1 A toy example of metric learning

Take the following toy example to demonstrate the effect of basic contrastive loss based approaches to the metric learning problem. Say we have measured three features of an “animal" dataset comprised of dog, cat, and snake data. We then plot the raw measured data in $\mathbb{R}^3$.

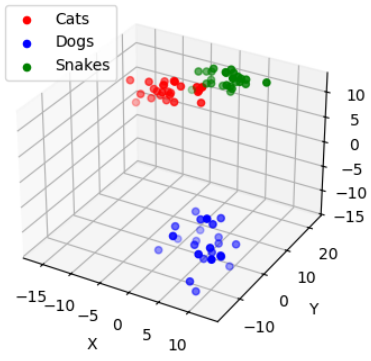

Figure 7: Standard metric learning technique clusters appropriately, but loses distinction between mammals and reptiles

Though these three classes should remain distinct, we should expect cat and dog points to be closer to each other than they are to snake points, by virtue of them both being mammals, having four legs, etc. Metric learning aims to learn a distance metric that brings cat and dog points close together, while keeping mammal and reptile points far apart.

Classic approaches to metric learning often involve some degree of supervision. In this example, say we are given supervision in the form of similar and dissimilar pairs: Given a training pair (x_1, x_2) and a label

$$Y = \begin{cases} 0, & (x_1, x_2) \notin S \\ 1, & (x_1, x_2) \in S \end{cases}$$

where S is the set of similar pairs defined as $(x_1, x_2) \in S$ if x_1, x_2 both mammals or x_1, x_2 both reptiles, define a training tuple as (Y, x_1, x_2). We can then define a contrastive loss penalty as

$$\mathcal{L}(Y, x_1, x_2) = Y \cdot d(x_1, x_2)^2 + (1 - Y) \cdot [\gamma - d((x_1, x_2)^2]_+ \tag{3}$$

where γ is a pre-defined margin parameter, $d(\cdot, \cdot)$ is the Euclidean norm, and $[\cdot]_+$ is the hinge function defined as $[\cdot]_+ = \max(\cdot, 0)$. We can see that such a loss function penalizes *large* distances between *similar* points, and *small* distances between *dissimilar* points. In other words, contrastive loss encourages small distances between similar points, and large distances between dissimilar points.

A contrastive loss distance metric learning algorithm could thus take the following form (the "Siamese network" architecture) [1].

Algorithm 1: Contrastive metric learning

Data: points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, set of similar pairs S, set of dissimilar pairs D, weight α
initialize a neural network with random weights
while *not converged* **do**
  for *pair* (x_i, x_j) *and label* Y **do**
    embed x_i, x_j through the NN
    calculate loss with equation (1)
    backpropagate
  end
end

Such a neural network is capable of learning a distance metric that separates the blue and red points from the green points. However, such an algorithm is incapable of maintaining separation between the red and blue points; since no part of the loss function penalizes the *mixing* of the red and blue (cat and dog) points, we lose any separation between these two clusters – even though the original embedding showed a clear distinction between the two. This example highlights the necessity of a

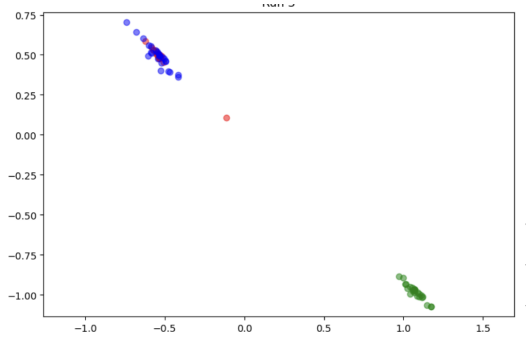

Figure 8: With only contrastive loss, cat and dog points are brought together, but the separation between the two is lost.

more sophisticated algorithm for distance metric learning that can keep the green points separate from the red and blue points, but also prevents the red and blue points from collapsing onto each other.

4.2 Contrastive loss with locality

We amend the contrastive loss function (1) to penalize the lack of local structure. Our new contrastive loss function will take the form

$$\mathcal{L}(Y, x_1, x_2) = \ Y \cdot d(x_1, x_2)^2 + (1 - Y) \cdot [\gamma - d((x_1, x_2)^2]_+ + \alpha f(\cdot) \tag{4}$$

where $f(\cdot)$ is an additional penalty that enforces local structure within the learned metric (with a weight term α). We aim to find a suitable function f that works with the contrastive loss function (e.g. is differentiable and well-behaved).

4.3 Defining the locality penalty

t-SNE is a dimensionality reduction algorithm that is capable of embedding high-dimensional data into low dimension while maintaining local structure [6]. It accomplishes this by calculating a "similarity matrix" for the original high-dimensional points and for the low-dimensional embedding, and then calculating how similar the low-dimensional similarity matrix is with the high-dimensional similarity matrix. t-SNE then performs gradient descent to minimize the KL divergence between the high and low dimensional embeddings. Unfortunately, t-SNE does not give a *metric*, e.g. after t-SNE calculates an embedding, if you want to embed two new points, you need to run the entire algorithm again; there is also no significance to cluster separation or distance in t-SNE embeddings [9]. Thus, we use t-SNE as a starting point for our algorithm, but operate under the broad frame of constrastive loss neural networks to learn a true embedding function.

Inspired by t-SNE, we use a high dimensional and low dimensional affinity matrix to model local structure. We first define our high dimensional affinities. For original points x_i, x_j, define

$$p_{j|i} = \frac{\exp\{(-||x_i - x_j||^2/2\sigma_i^2\})}{\sum_{k \neq i} \exp\{(-||x_i - x_k||^2/2\sigma_i^2\})}$$

where σ_i is found via binary search identically to in t-SNE (σ_i can be thought of as a "neighborhood size" parameter). We then define the high dimensional affinity matrix P, with the i, j-th element defined as

$$P_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Similarly, once the points are embedded into some new space, we define the low dimensional affinity matrix as

$$Q_{ij} = \frac{\exp\{(-||y_i - y_j||^2\})}{\sum_{k \neq l} \exp\{(-||y_k - y_l||^2\})}$$

where y_i, y_j are the embedded points. Finally, recall that the KL divergence between P and Q is given as

$$C_{KL} = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

where P_i is the conditional probability distribution over all points given a point x_i and Q_i is the conditional probability distribution over all embedded points given an embedded point y_i. In this way, the KL divergence measures how faithfully the low dimensional embedding matches the original high dimensional affinities.

Given this setup, we learn a metric via the following algorithm.

Algorithm 2: Contrastive metric learning with KL divergence

Data: points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, set of similar pairs S, set of dissimilar pairs D, weight α
calculate high dimensional affinities P_i between all points in S
while *not converged* **do**
  for *pair* (x_i, x_j) *and label* Y **do**
    embed x_i, x_j through the NN
    recalculate the low dimensional affinity matrix Q between all points in S given the new embedding from the NN
    calculate loss with equation (2) with $f(\cdot) = C_{KL}(P, Q)$
    backpropagate
  end
end

We design slightly different neural network architectures depending on the type of input data; for instance, if the input data is an image, we prefer to use a convolutional neural network instead. The architectures used in our experiments are the following.

For non-image data, we employ a six layer fully connected neural network with dimension $d - 128 - 128 - 64 - 64 - 2$ where d is the dimension of the input data and ReLU activation between each FC layer.

For image data, we employ a convolutional neural network with 6 layers total, comprising of 3 convolutional layers and 3 fully connected layers. The 3 convolutional layers have filters (16, 32, 64), each using 3x3 kernels, followed by a BatchNorm, ReLU activation, and 2x2 max pooling layer. Subsequent fully connected layers have shape $64 \times 3 \times 3 \to 128 \to 64 \to 2$, with ReLU activation between each layer.

4.4 Experimental Results

We evaluated the effectiveness of this algorithm on the same CIFAR-10 dataset. We used a number of different metrics to evaluate the improvement in clustering output.

Define the discounted cumulative gain loss as follows. Let

$$\text{rel}(x, y) = \begin{cases} 1, & \text{if } x \text{ and } y \text{ belong to the same fine-grained class} \\ 0.5, & \text{if } x \text{ and } y \text{ belong to different fine-grained classes but the same coarse class} \\ 0, & \text{otherwise} \end{cases}$$

Then the DCG loss is defined as the difference between the "ideal" ordering of neighbors compared with the learned ordering of neighbors, where neighbors are ordered by distance, and neighbors with similar labels should be higher on the ordinal ranking for a given point.

$$\text{DCG}(x_q) = \sum_{i=1}^{n} \frac{2^{\text{rel}(x_q, x_i)} - 1}{\log_2(i + 2)}$$

We can also define a context-sensitive loss as

$$\text{CSL}(y, \hat{y}) = \begin{cases} 0, & y = \hat{y} \\ \text{LCA}(y, \hat{y}), & y \neq \hat{y} \end{cases}$$

where $\text{LCA}(y, \hat{y})$ is the height of the lowest common ancestor of labels y, $\hat{y}$ on a hierarchical clustering dendrogram normalized to height 1. This provide a more nuanced alternative to $0 - 1$ loss that incorporates partial credit for misclassifications. Total CSL error is defined as the overall $\%$ classification error using the CSL penalty.

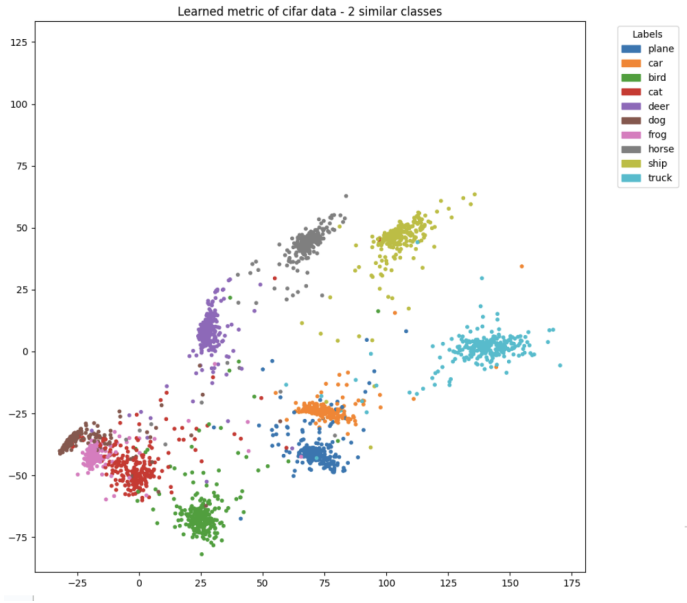

Figure 9: Metric learning with the labels for 'car' and 'plane' combined, as well as 'frog' and 'dog'. Our weak metric learning algorithm can bring the plane and car points close together, but does not collapse the separation between the two. Likewise for the frog and dog points.

The learned metric has nominal improvements on all three metrics compared to data learned without metric learning, indicating an improvement of substructure preservation with the KL divergence approach.

CIFAR DATA (2 similar classes, 2000 test points)			
	nDCG	**Average CSL Penalty**	**Total CSL Error**
Before metric learning	0.9821028259	0.9262126291	0.0839081076
After metric learning	0.9906358107	0.6085862296	0.0828510146

5 Conclusions and Future Work

We believe that the joint learning approach has a lot of potential as a way to maintain the substructure of classes when only given a coarse set of labels. The initial experiments done indicate a non-negligible improvement on neighborhood purity on the order of 10%. Further optimizations could be considered to improve this number further, e.g. an improved balancing of the base loss and KL loss terms, dynamic adjustment of the choice of C, or a combination of thresholding or L^2 penalty on the KL loss.

6 Bibliography

[1] Chopra, S., Hadsell, R. & LeCun, Y. (2005) Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1, pp. 539–546. IEEE.

[2] LeCun, Y., Huang, F.J. & Bottou, L. (2004) Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, pp. II-104. IEEE.

[3] Linderman, G.C., Rachh, M., Hoskins, J.G., Steinerberger, S. & Kluger, Y. (2019) Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data. *Nature Methods* **16**(3):243–245.

[4] van der Maaten, L. (2009) Learning a Parametric Embedding by Preserving Local Structure. In D. van Dyk & M. Welling (eds.), *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, Vol. 5, pp. 384–391. PMLR.

[5] van der Maaten, L. (2013) Barnes-Hut-SNE. *arXiv preprint arXiv:1301.3342*. URL https://arxiv.org/abs/1301.3342

[6] van der Maaten, L. & Hinton, G. (2008) Visualizing Data using t-SNE. *Journal of Machine Learning Research* **9**(86):2579–2605.

[7] Xing, E., Jordan, M., Russell, S.J. & Ng, A. (2002) Distance Metric Learning with Application to Clustering with Side-Information. In S. Becker, S. Thrun & K. Obermayer (eds.), *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.

[8] Xie, J., Girshick, R. & Farhadi, A. (2016) Unsupervised Deep Embedding for Clustering Analysis. In M.F. Balcan & K.Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48, pp. 478–487. PMLR.

[9] Zhou, Y. & Sharpee, T.O. (2018) Using global t-SNE to preserve inter-cluster data structure. *bioRxiv* 331611. doi:10.1101/331611.