APPA: A Cluster-Preserving Approximating Parametric Projection Algorithm

Alister Machado[1][a] and Alexandru Telea[1][b]
[1]*Department of Information and Computing Sciences, Utrecht University, The Netherlands*
{*a.machadodosreis, a.c.telea*}*@uu.nl*

Keywords: Dimensionality Reduction, Neural Networks, Parametric Projections.

Abstract: Dimensionality Reduction (also called projection) is the tool of choice for visualizing high-dimensional data due to its applicability to datasets of different kinds, sizes, and dimensionalities. However, many projection algorithms scale poorly with dataset size, limiting their applicability on large datasets. A particularly successful approach to scalability is to approximate the projection by computing it for a subset of the data, and use the (fast) approximation to project the full dataset. Neural Network Projection (NNP) is one such approximation algorithm which is fast at both training and inference, is projection- and dataset-agnostic, has out-of-sample ability, and is simple to implement. Yet, NNP creates projections in which data points diffuse over the projection space even if they were clustered at training time. Since groupings are crucial features of projections this severely limits NNP's attractivity as an alternative projection technique. We propose APPA (Approximating Parametric Projection Algorithm), a refinement of NNP that inherits all of NNP's qualities while strongly reducing the diffusion problem. We evaluate APPA across a variety of datasets and projection techniques, demonstrating its ability to maintain the quality of the reference projection.

1 INTRODUCTION

Exploratory data analysis is an increasingly common task given the widespread use of data collection techniques in the most varied domains. Visually exploring data that has a large number of attributes, also called dimensions, per measurement (also called sample) calls for the development of specific techniques. In this context, projection – also called Dimensionality Reduction (DR) – algorithms support exploratory data analysis by transforming data-space tasks such as *measuring* correlations, relationships, and naturally-occurring clusters, into *visual* tasks. They do so by mapping the data from its original data space to a lower-dimensional space (typically 2D or 3D). The data is then visualized by means of a scatterplot, giving rise to a projection plot in which visual patterns can be used to infer properties of the data.

Tens of DR methods exist with readily available implementations in public software packages. At a high level, one can group these into linear and global methods such as Principal Component Analysis (PCA) (Pearson, 1901), non-linear global methods such as Multidimensional Scaling (MDS) (Kruskal, 1964a), and non-linear local methods such as t-SNE (van der Maaten and Hinton, 2008) and UMAP (McInnes et al., 2020). Linear and global methods are simple to implement and fast to execute but can only preserve a limited range of data patterns in the resulting projections. Non-linear local methods are typically more *expressive* than their linear (and/or global) counterparts, *i.e.*, they have more *flexibility* as to where they project each data point. This comes at a cost in time and/or space complexity though. The computation of the distance matrix and pairwise interactions for t-SNE, or the creation of a graph approximating the data manifold in UMAP are costly steps in each respective algorithm. Additionally, both of these methods (and several others in the same class) are *non-parametric* – they require the complete recalculation of a projection when new data points arrive or existing points change values. Given this high complexity, avoiding their complete recalculation introduces significant performance benefits.

While parametric versions of specific algorithms have been proposed, such as Parametric t-SNE (van der Maaten, 2009), an *algorithm-agnostic* solution allows one to use *any* DR algorithm to new data and also to speed up the projection computation. To do this, one would apply the desired DR algorithm to a small subset of the data (which is fast) and then use a fast parametric

[a] https://orcid.org/0000-0002-1129-4628
[b] https://orcid.org/0000-0003-0750-0502

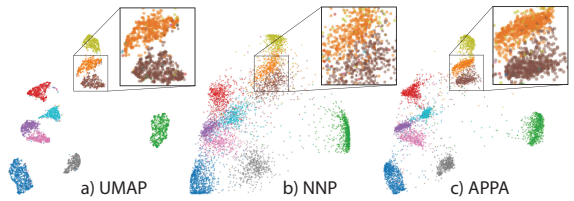

Figure 1: Diffusion effect of NNP. a) Projection of the USPS dataset created by UMAP. b) The approximate projection created by NNP. Diffusion smooths out well-defined clusters in (a) and merges neighboring clusters. c) Our technique, APPA, shows reduced diffusion and well-separated clusters.

approximation to project the remaining data. A recent algorithm capable of doing so is NNP (Neural Network Projection) (Espadoto et al., 2020b). This algorithm builds a Neural Network approximating any projection P by minimizing a regression error.

NNP enjoys a set of properties that make it interesting for projecting large datasets. It is *projection-agnostic*, meaning it does not place any requirements on the projection P which it approximates. Its backbone is a deep neural network, so it inherits the desirable properties of those: training is *fast* and can be done on the GPU using any off-the-shelf gradient-based optimizer. Inference scales *linearly* with the number of data points, as opposed to *e.g.* UMAP and t-SNE, which scale super-linearly. Additionally, NNP builds a function that works on *unseen data* (also called out-of-sample data), avoiding costly recomputation. NNP's implementation is *simple* and requires *no parameters* to be tweaked with in inference mode. Last but not least, NNP's neural network can be adapted depending on the data type one wishes to project – *i.e.*, convolutional networks for time series data or images and attention networks for text data.

However, NNP has a key drawback: Its projection function causes well-defined point clusters in the training projection to spread out, or diffuse, over the projection space, reducing the inter-cluster distance and mixing different data categories (see Fig. 1). This is highly undesirable since it conveys a far poorer cluster separation than the actual training projection captures from its input data. The diffusion effect *worsens* when projecting out-of-sample data – while this is the key use-case for NNP. All in all, diffusion negates all other desirable properties of NNP and makes it unable to replace slower, but higher-quality, projection techniques such as t-SNE or UMAP.

In this work, we propose APPA, standing for **A**pproximating **P**arametric **P**rojection **A**lgorithm, a technique that strongly reduces the diffusion problem of NNP. APPA does so by using regularization techniques with the help of soft barrier functions, with only $O(n)$ extra computational cost in its training loop for n data points and *no additional cost* at inference time. Its structure is otherwise identical to NNP, meaning it inherits all of NNP's desirable properties, *e.g.*, genericity, speed, out-of-sample ability, and ease of implementation and use. We make APPA's Python code openly available (Machado and Telea, 2025a).

The structure of this paper is as follows. Section 2 covers related work with a focus on approximating methods for projection. Section 3 details both NNP and how we extend to it that tackle NNP's diffusion problem. Section 4 evaluates our method against several well-known projection methods, including NNP, on a variety of datasets and shows our method's ability to compute high-quality, low-diffusion, projections. Section 5 discusses our method's key features and concludes the paper with future work directions. We provide additional information and experimental data in the supplemental material (Machado and Telea, 2025b).

2 RELATED WORK

Notations: We denote by $\mathbf{X}$ a dataset containing n data points $\mathbf{x}_i \in \mathbb{R}^d$ where $\mathbf{x} = (x_1, \ldots, x_d)$. The projection of a single data point is denoted by $\mathbf{y}_i = P(\mathbf{x}_i)$, where $P : \mathbf{X} \to \mathbb{R}^q$ is any DR algorithm; the projection of the entire dataset is denoted by $\mathbf{Y} = P(\mathbf{X})$. P is computed so as to minimize various types of costs, or errors, between $\mathbf{X}$ and $P(\mathbf{X})$. A neural network used to approximate P is denoted by $\Pi_\phi : \mathbb{R}^d \to \mathbb{R}^q$, where ϕ are weights and bias parameters. We denote the Euclidean norm, respectively L1 norm, of a vector by $\|\mathbf{x}\|_2 = (\sum_j x_j^2)^{\frac{1}{2}}$, respectively by $\|\mathbf{x}\|_1 = \sum_j |x_j|$.

2.1 Dimensionality Reduction

Tens of DR algorithms exist, each with different properties and mathematical underpinnings. For example, PCA (Pearson, 1901) is a linear parametric method that minimizes the so-called reconstruction error over a dataset. MDS (Kruskal, 1964a; Kruskal, 1964b) computes P so as to minimize the differences in normalized pairwise distances in $\mathbf{Y}$ *vs* those in $\mathbf{X}$. t-SNE (van der Maaten and Hinton, 2008) and UMAP (McInnes et al., 2020) are non-linear and non-parametric methods that approximate local point-neighborhood structures in the data space and map them to the projection space. Such methods are described in further detail in a range of surveys (Espadoto et al., 2019; Nonato and Aupetit, 2018; Sorzano et al., 2014).

Non-parametric methods allow for greater flexibility during the optimization of their (often complex) cost functions. For example, in t-SNE (van der Maaten and Hinton, 2008), Gaussian probability distributions with varying scales are used to model the data in $\mathbb{R}^d$.

Those are mapped to Student t distributions in the projection space by iterative gradient descent on their KL divergence – a measure of dissimilarity between probability distributions. This effectively creates a force-directed placement, where each point in the projection can *individually* move in space to find its best projection spot. This is not true for parametric methods: Since they compute a *single function* $P : \mathbb{R}^n \to \mathbb{R}^q$ that is applied to every data point to compute its projection, as opposed to a mapping $P : \mathbf{X} \to \mathbb{R}^q$, altering the position of one point typically affects other points as well.

Parametric methods have several key advantages, however. First and foremost, they can compute the projection of *new*, *unseen* data points – they effectively project the entire space $\mathbb{R}^n$ rather than a specific fixed set $\mathbf{X} \subset \mathbb{R}^n$. This allows one to construct P based on data $\mathbf{X}$ existing at some point in time and next augment the result with new points coming from the same distribution without having to recompute the entire projection (which would be slow but could also cause spurious changes in the resulting scatterplot). Separately, a parametric method can be fit on a subset $\mathbf{X}_s \subset \mathbf{X}$ of the data, where $|\mathbf{X}_s| \ll |\mathbf{X}|$. The learned parameters can then be used to project the entire dataset $\mathbf{X}$. Using a representative subsample $\mathbf{X}_s$ yields typically small approximation errors. Projecting the entire dataset $\mathbf{X}$ can then be done using the learned parameters, providing major performance improvements (Espadoto et al., 2020b).

2.2 Projecting Unseen Data

The advantages of parametrized projection algorithms have led to numerous such techniques. We next highlight a few examples relevant to our work.

Parametric t-SNE (van der Maaten, 2009) uses a multi-staged pipeline to build an approximation that preserves the local structure present in a computed t-SNE projection. This approach accurately extends t-SNE to a parametric setting but is computationally expensive; is complex to implement; and requires careful setting of many hyperparameters.

PCA and NCA (Goldberger et al., 2004) are (by design) parametric approaches. They compute linear transformations $\mathbf{X} \mapsto \mathbf{XW}$, where $\mathbf{W}$ can also be applied to unseen data. However, their expressiveness is limited by their linear nature.

Auto-Encoders (AEs) (Hinton and Salakhutdinov, 2006) can be used to learn projections by setting the number of neurons in the bottleneck of the network to q. The Encoder creates a projection to $\mathbb{R}^q$ and the Decoder outputs a reconstruction to $\mathbb{R}^d$. As plain AEs are typically poor at capturing local relationships and showing good class separation in projections, several extensions thereof have been proposed (Makhzani et al., 2015; Espadoto et al., 2021; Machado et al., 2024). These rely, among others, on pseudo-labels inferred from clustering or use Generative Adversarial Networks (GANs) to increase the desired class separation in the resulting projections.

As it is hard to design an algorithm that can produce a parametrized, high-speed projection without incurring significant quality penalties, alternative approaches aim to *mimic* reference projections. Neural Network Projection (NNP) (Espadoto et al., 2020b) learns to mimic any given reference projection $P(\mathbf{X})$ by fitting a feed-forward neural network with a regression target. This method suffers from what we claim is overfitting (see next Sec. 3) which manifests visually as *diffusion* when the learned projection is applied to unseen data (see Fig. 1). Attempts to alleviate this problem, for example learning projections of neighborhoods instead of single points (Modrakowski et al., 2022), or tweaking the training hyperparameters (Oliveira et al., 2023; Espadoto et al., 2020a), have shown only very limited success – the diffusion phenomenon is still visible in their results.

We present a principled justification as to why NNP degrades in the out of sample regime, and our new APPA algorithm that yields a measurable quality improvement in terms of learning to approximate any given reference multidimensional projection. We further develop a sample-based version of APPA, allowing control over the amount of desirable diffusion introduced in the learned projection.

3 METHOD

Looking again at Fig. 1, we see that in dense regions of the projection space, NNP is reasonably accurate, so we do not want to strongly modify it there. Between clusters, however, NNP degrades and introduces diffusion. As such, we design our method APPA to *regularize* NNP by avoiding placing points in low-density regions in the scatterplot, as described next.

3.1 Neural Network Projection (NNP)

(Espadoto et al., 2020b) first proposed to use a neural network $\Pi_\phi : \mathbb{R}^n \to \mathbb{R}^q$ to parametrically approximate any reference projection computed on some dataset $\mathbf{X}$ by a DR technique of choice P. The idea is simple and quite effective: Given training data $\mathbf{Y} = P(\mathbf{X})$, NNP learns the parameters ϕ by optimizing the mean squared error loss

$$\mathcal{L}_{\text{NNP}}(\phi) = \sum_{i=1}^{n} \|\mathbf{y}_i - \Pi_\phi(\mathbf{x}_i)\|_2^2. \tag{1}$$

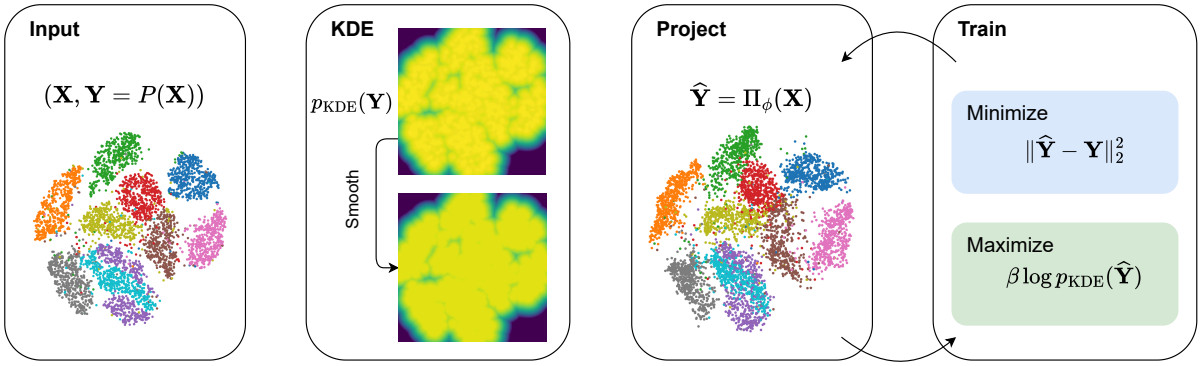

Figure 2: APPA's architecture. APPA takes as input a dataset and its projection, computed by any method. APPA then builds a smooth KDE which determines the low (dark blue) and high (yellow) density areas in the projection. The approximating neural network is then fit to the reference projection with the additional goal of avoiding low-density areas. See Section 3.2.

between the network's predictions and the reference projection $\mathbf{Y}$.

NNP additionally uses *early stopping*, a simple form of regularization. Once the error on a held-out validation set stops decreasing, training is stopped. While valid, this technique is not enough to prevent overfitting, which is responsible for low projection quality when using the trained network Π_ϕ on unseen data. This quality loss is reflected both by lower quality metrics and visually by the 'diffusion' of point clusters that are well separated in the reference projection.

k-NNP (Modrakowski et al., 2022) aims to improve NNP by learning to project *neighborhoods* of points from the data space to the projection space. This is motivated by the observation that projections are inherently non-local: the position of each point depends on the position of possibly every other point in the projection – with influence diminishing according to distance between such points. While *k*-NNP marginally improves on NNP's quality, it still fails to prevent the mentioned diffusion problem. Separately, a wealth of training heuristics, hyperparameters, and loss functions have been explored with NNP (Oliveira et al., 2023; Espadoto et al., 2020a). As for *k*-NNP, these only yielded marginal quality improvements.

3.2 APPA: Avoiding Low-Density Areas

NNP's training procedure consists of fitting a regression model from $\mathbb{R}^d$ to $\mathbb{R}^q$. This can be understood as finding the maximum likelihood estimate (MLE) under a Gaussian conditional model

$$p_\phi(\mathbf{y}|\mathbf{x}_i) = \mathcal{N}(\mathbf{y}|\mu = \Pi_\phi(\mathbf{x}_i), I) \quad (2)$$

$$\hat{\phi} = \arg\max_\phi \log p_\phi(\mathbf{y}_i|\mathbf{x}_i) \propto \|\mathbf{y}_i - \Pi_\phi(\mathbf{x}_i)\|_2^2, \quad (3)$$

where the maximization is performed over the training set $(\mathbf{X}, \mathbf{Y} = P(\mathbf{X}))$.

While this works for learning *point-wise* relationships between data points $\mathbf{x}_i$ and their projections $\mathbf{y}_i$, it fails to consider how the projection algorithm P induces a *probability distribution* over the projection space. The overall effect of P can be taken into account by introducing a *prior* and resorting to Maximum A Posteriori (MAP) estimation

$$\hat{\phi} = \arg\max_\phi \log p(\mathbf{y}|\Pi_\phi(\mathbf{x_i})) + \log p(\Pi_\phi(\mathbf{x}_i)). \quad (4)$$

Note that NNP uses only the first term in the right hand side of Eqn. 4. The second term – that we introduce with APPA – constrains the projection, sharpening its shape by explicitly avoiding low-probability regions.

The choice of an appropriate prior is crucial for APPA's success. We use a prior derived by Kernel Density Estimation (Rosenblatt, 1956), $p(\mathbf{y}) = \mathrm{KDE}(\mathbf{Y})$, computed over the reference (training) projection $\mathbf{Y} = P(\mathbf{X})$. We use a Gaussian kernel with small bandwidth (0.01) since we want our prior to quickly decay away from the training projection. While non-parametric, this prior is differentiable with respect to its argument, so it is suitable for use in APPA's loss function. We also replace NNP's Gaussian likelihood by a Laplacian likelihood to further strengthen the desired sharpening effect. This amounts to replacing NNP's Mean Squared Error loss with a Mean Absolute Error (L1) loss.

Putting all above together, APPA's loss function to be minimized is

$$\begin{aligned}\mathcal{L}_{\mathrm{APPA}} &= \sum_{i=1}^{n} \left[-\log p_\phi(\mathbf{y}_i|\mathbf{x}_i) - \beta \log p(\Pi_\phi(\mathbf{x}_i)) \right] \\ &= \sum_{i=1}^{n} \|\mathbf{y}_i - \Pi_\phi(\mathbf{x}_i)\|_1 - \beta \log p(\Pi_\phi(\mathbf{x}_i)), \quad (5)\end{aligned}$$

where the first term in the summation is a regression error from the reference projection, and the second is a regularization term with weight β encouraging the distribution of the approximating projection to match that of the reference projection. Simply put, our training combines two objectives: in high-density regions, we

reproduce the reference projection; in low-density regions, we regularize towards the reference projection's shape. In all our experiments, we use $\beta = 0.002$.

To allow training to focus only on faithfully reproducing the projection in high-density areas, we clip the log-prior values to a maximum of -2. This flattens high prior probability areas causing the gradient of the regularization term to vanish there. For this we use

$$\mathcal{L}_{\text{APPA}} = \sum_{i=1}^{n} \|\mathbf{y}_i - \mathbf{p}_i^{\phi}\|_1 - \beta \min(\log p(\mathbf{p}_i^{\phi}), -2), \quad (6)$$

where we denote $\mathbf{p}_i^{\phi} = \Pi_{\phi}(\mathbf{x}_i)$.

As Sec. 4.2 next shows, our loss function (Eqn. 6) produces slightly worse results than NNP's loss (Eqn. 1) on the training set, as expected given our extra regularization – but strongly reduces diffusion on unseen data.

3.3 Sampled APPA

As explained in Sec. 3.2, APPA's regularization term (Eqn. 6) introduces an effect opposite to diffusion, which we call *sharpening*. In practice, users may want to control the strength of this sharpening effect. This is difficult to do reliably in APPA. For this reason, we present next a Sampled version of APPA (SAPPA), which allows simple sharpening control.

SAPPA achieves the same effect as APPA – *i.e.*, avoid low-density areas – by sampling the projection space regions we would like to avoid and then explicitly penalizes closeness to such samples – see Fig. 3.

For this, we first compute a *barrier function* $\mathcal{B}$: $\mathbb{R}^q \to \mathbb{R}$ that combines the effect of two terms as follows. The first term $Z(\mathbf{y})$ disables the barrier close to points in the training projection $\mathbf{Y}$ and enables it slightly further away from such points; this prevents diffusion close to points in $\mathbf{Y}$. We compute Z as

$$Z(\mathbf{y}) = \frac{1}{1 + p(\mathbf{y})}, \quad (7)$$

where p is the KDE-based prior introduced in Sec. 3.2 and $\mathbf{y}$ is any point in the projection space $\mathbb{R}^q$.

The second term disables the barrier at points $\mathbf{y}$ that are far away from *all* points in the reference projection. This term is defined as

$$D(\mathbf{y}) = \frac{1}{1 + \min_{\mathbf{y}' \in \mathbf{Y}} \|\mathbf{y} - \mathbf{y}'\|}, \quad (8)$$

which we compute by using a nearest-neighbor algorithm. The rationale behind using D is that we do not want to explicitly constrain our projection in areas far away from the training projection $\mathbf{Y}$.

The final barrier is computed by multiplying the two desired effects as

$$\mathcal{B}(\mathbf{y}) = Z(\mathbf{y}) D(\mathbf{y}) \quad (9)$$

Table 1: Datasets used for evaluation in this work. †: post-processed subsets of original datasets. See supplemental material for additional details.

Name	Dims. (d)	Size (n)	# Classes
Cats and Dogs	2048	25 000	2
CIFAR10†	1920	12 000	2
CIFAR100†	1920	6 000	2
FashionMNIST	784	60 000	10
HAR	561	7 352	6
IMDB	500	25 000	2
MNIST	784	60 000	10
Reuters	5000	8 432	6
Spambase	57	4 601	2
USPS	256	9 290	10

We uniformly sample $\mathcal{B}$ on a 300×300 grid and keep only the 80% highest values, creating a set of barrier points $B = \{\mathbf{b}_j\}$. Using these barrier points, we construct the loss

$$\mathcal{L}_{\text{SAPPA}} = \sum_{i=1}^{n} \|\mathbf{y}_i - \mathbf{p}_i^{\phi}\|_1 + \beta \sum_j (\delta - \|\mathbf{p}_i^{\phi} - \mathbf{b}_j\|_2^2)_+, \quad (10)$$

where $\mathbf{p}_i^{\phi} = \Pi_{\phi}(\mathbf{x}_i)$ and $(x)_+ = \max(0, x)$. Simply put, this loss adds a linear penalty to points within a band of width δ from the training projection $\mathbf{Y}$, so limits diffusion close to $\mathbf{Y}$; and has no effect further away from $\mathbf{Y}$. Users can then control δ to tune the desired sharpening effect. In our experiments, we fix $\beta = 1.0$.

4 EVALUATION

We evaluate our approach for a variety of projection methods P – namely t-SNE, UMAP, Isomap, and PCA – across 10 datasets commonly present in DR literature benchmarks (see Tab. 1). We compare our results with NNP (Sec. 4.1) and see how (S)APPA strongly alleviates the diffusion problem. We further show that APPA yields better quality as measured by standard projection quality metrics (Sec. 4.2). Next, we explore how users can control sharpening via the parameter δ (Sec. 4.3). We also explore how (S)APPA behaves when projecting increasing amounts of data unseen during training (Sec. 4.4). Finally, we show how our APPA computationally scales to handle projecting large datasets (Sec. 4.5).

4.1 Generating projections

We design our evaluation as follows. Each combination of dataset $\mathbf{X}$ and projection P makes an experiment dataset $\mathbf{D} = (\mathbf{X}, \mathbf{Y} = P(\mathbf{X}))$. We split $\mathbf{D}$ into a training set $\mathbf{D}_T = (\mathbf{X}_T, \mathbf{Y}_T)$ and an evaluation set $\mathbf{D}_E = (\mathbf{X}_E, \mathbf{Y}_E)$, with $|\mathbf{D}_T| = \frac{1}{3}|\mathbf{X}|$ and $|\mathbf{D}_E| = \frac{2}{3}|\mathbf{X}|$.

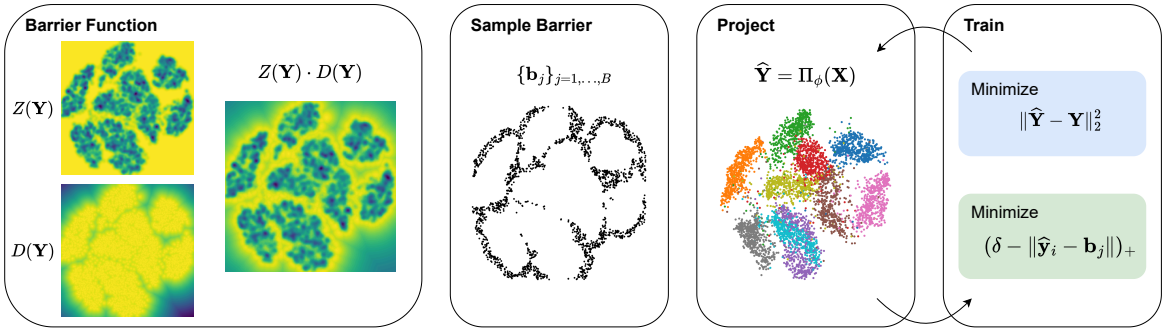

Figure 3: SAPPA's architecture. This version of APPA uses an explicit set of barrier points $\mathbf{b}_j$. The algorithm minimizes the regression error with respect to the reference projection $\mathbf{Y}$ as well as a penalty linear in the distance of all points to the barrier. See Section 3.3 for further details on sampling the barrier and the proximity penalty.

We then train the approximation algorithms NNP, APPA, SAPPA on $(\mathbf{X}_T, \mathbf{Y}_T)$ for at most 300 epochs using the Adam optimizer (Kingma and Ba, 2015), compute their output $\widehat{\mathbf{Y}} = \Pi_\phi(\mathbf{X}_E)$, and compare it to the unseen part of the projection $(\mathbf{Y}_E)$.

Figure 4 shows the resulting projections (additional examples are present in the supplemental material). We immediately see diffusion arising when using NNP, both on the training data $\mathbf{X}_T$ and, even more so, on the test data $\mathbf{X}_E$. For example, consider NNP's output when trained on the t-SNE projection of the FashionMNIST dataset (Fig. 4, row 3 from top). In the reference projection, we see well-isolated sample groups such as the orange ones (A1) or yellow ones (B1). These strongly diffuse to mix with the other sample groups for both the training and testing NNP projection (markers A2, A3, B2, B3 in the image). For the (S)APPA training projection, these samples re-create the t-SNE training projection patterns almost exactly (markers A4, A6, B4, B6). For the (S)APPA testing projection, these groups become slightly more diffuse but still separate clearly from the adjacent groups in the projection (markers A5, A7, B5, B7).

(S)APPA's ability to contain diffusion is even more visible when learning UMAP, a DR method known to produce *sharp* separation – small dense clusters separated by a lot of white space. For the HAR dataset (Fig. 4, row 2 from top), the reference projection shows two very clearly separated clusters of cyan samples (C1) and yellow-and-pink samples (D1). These clusters exhibit very strong diffusion in both NNP training and testing projections (markers D2, D3, C2, C3). (S)APPA's training projection keeps these clusters almost identical to the reference projection (markers C4, D4, C6, D6) and creates minimal diffusion for the testing projection (markers C5, D5, C7, D7).

The trend of (S)APPA producing less diffusion than NNP, *i.e.*, better preservation of finer structures from the reference projections, holds across every single test case. Evidently, (S)APPA does not *completely* prevent the diffusion problem, for reasons already explained when outlining our design in Sec. 3: There is still a small amount of data that gets placed in regions that were *not* populated by the reference projection. This is due to the continuous nature of Π_ϕ: It is always possible to construct an $\mathbf{x}'$ such that it falls in an inter-cluster location in the learned projection. Separately, we see how (S)APPA cannot create *additional* separation which was not present in the reference projection – see the Cats & Dogs dataset projected with PCA in Fig. 4. Since PCA cannot separate well the two clusters (cyan and brown points) due to its linearity, neither NNP nor (S)APPA can do this. Yet, even in this case we see that (S)APPA yields less diffusion than NNP.

4.2 Projection Quality Metrics

As is standard practice when evaluating projections, we further back up the visual evidence outlining (S)APPA's higher quality in mimicking the reference projection (Sec. 4.1) by computing projection quality metrics for the compared methods and datasets. We use a total of 15 quality metrics, all common in DR literature, implemented by the library described in (Machado et al., 2025). For all metrics that operate on k-neighborhoods, we use $k = 51$. We next discuss a subset hereof; all remaining ones are detailed in the supplementary material.

True Neighbors Rate: Measures, for every projected point $\mathbf{y}_i$, the fraction of its k-nearest neighbors (KNN) that match the k-nearest neighbors of $\mathbf{x}_i$, its pre-image through P or Π_ϕ (Martins et al., 2014).

Trustworthiness and Continuity: Measure the amount of false neighbors (resp. missing neighbors) of each projected point, penalizing proportionally to the rank of the introduced (resp. missing) neighbor (Venna and Kaski, 2006).

Distance Consistency: This supervised cluster separation metric yields high values for projections in which each projected point is closer to the centroid

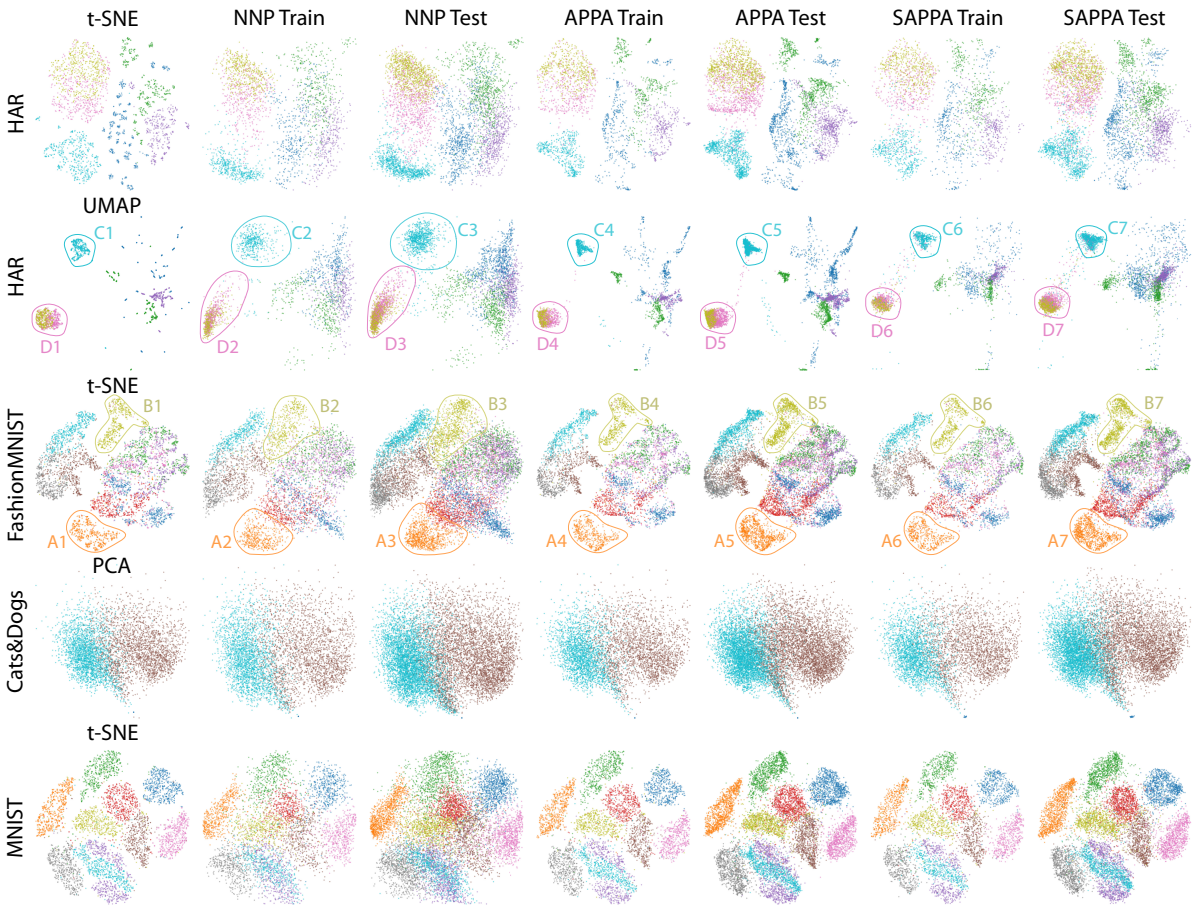

Figure 4: Using (S)APPA to reproduce different projections of different datasets. For each row we show, from left to right: the projection used for training; the result of running NNP on the training data and next on unseen data; APPA's output on the training data and next on unseen data; and finally, SAPPA's output on training data and next on unseen data. Overall, we see that (S)APPA better captures the training projection, and creates sharper clusters, than NNP.

of the points of its own class than to any other class centroid (Sips et al., 2009).

Neighborhood Hit: Counts the fraction of points around $\mathbf{y}_i$ that have the same class as $\mathbf{y}_i$ (Paulovich et al., 2008).

Stress: Measures the amount of pairwise distance distortions induced by Π_ϕ or P, taking as ground truth the pairwise distances $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ (Kruskal, 1964a; Smelser et al., 2024).

Figure 5 shows these metrics computed for all reference projections and datasets described in Sec. 4.1. In general, the reference projection method yields best values as expected – an *approximation* method, like NNP or (S)APPA, will likely not exceed this baseline. (S)APPA metrics come next, practically equal to each other and just slightly lower than the reference values. NNP metrics are consistently lower than (S)APPA ones indicating what we qualitatively saw so far, *i.e.*, that (S)APPA is a better approximator. Indeed, if diffusion is reduced, then a projected point's neighbors are more often correct (True Neighbors, Continuity, Trustworthiness); those neighbors more often share the same class (Neighborhood Hit); and distances are better preserved (Stress). Conversely, if a projection becomes diffuse, like in the NNP case, then neighborhoods break apart and/or lose accuracy (impacting True Neighbors, Continuity, Trustworthiness). Diffusion also causes points that should be close together to shift away from each other. If the reference projection had good Neighborhood Hit, this shifting will lower its value, since previously well-defined neighborhoods start mixing. Distance Consistency also goes down in diffuse projections: If classes spread around the projection space, their centroids tend to be potentially close to each other. Finally, diffusion also increases Stress, as points become more uniformly distributed in the projection space, something that unlikely happens in data space due to the curse of dimensionality.

Figure 6 shows the accuracy of the approximate projections *vs* the projection they are trained to approx-

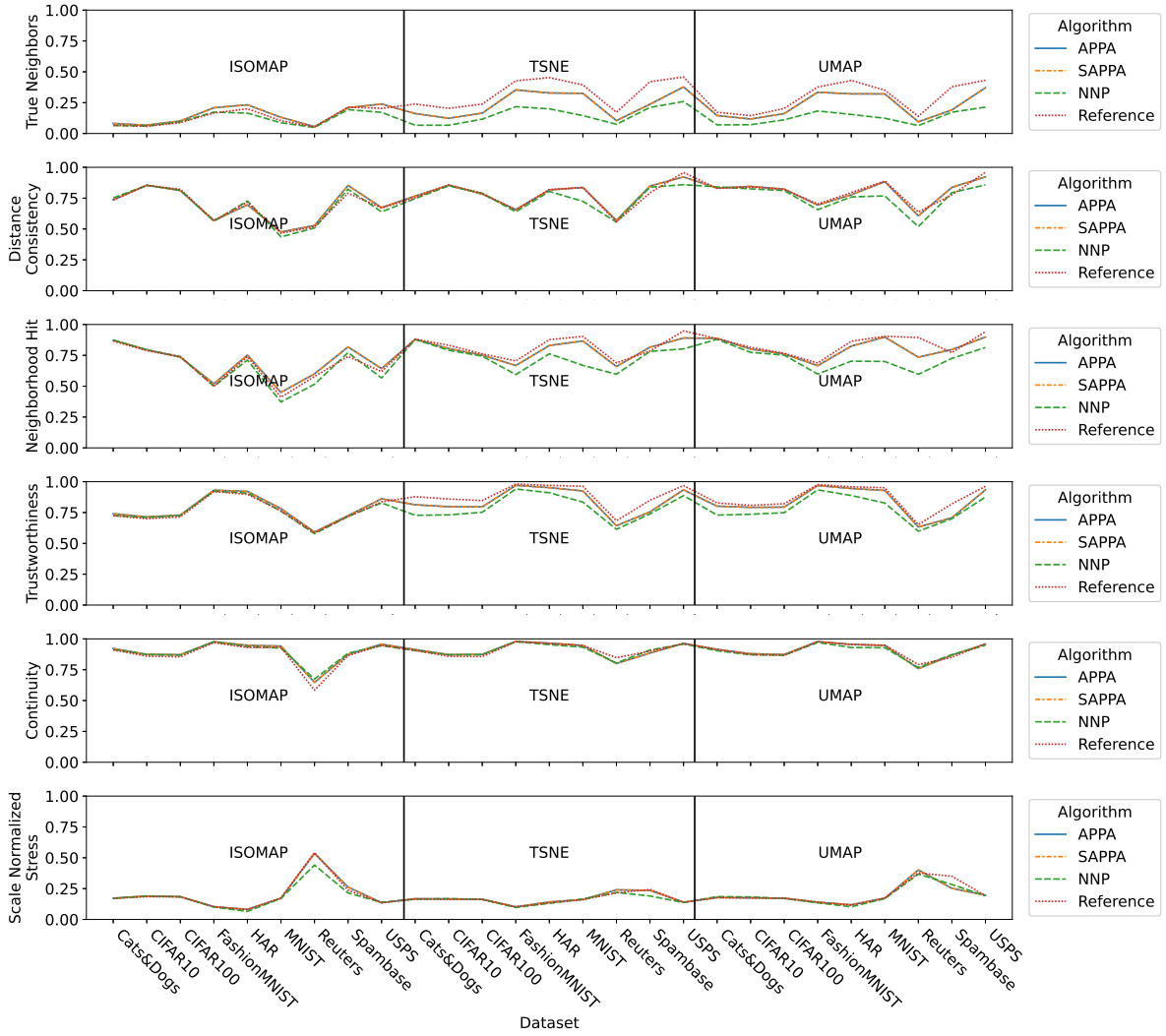

Figure 5: Projection quality metrics for (S)APPA and NNP when approximating different projection techniques. 'Reference' gives the value of each quality metric for the approximated projection. We see a clear "sandwiching" pattern: the reference projection has best quality metric values, followed by APPA, then by NNP.

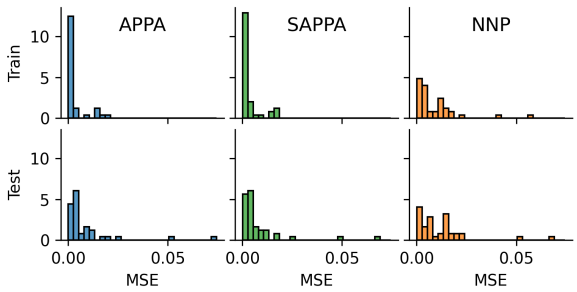

Figure 6: Mean Squared Error measured over training and test data for APPA, SAPPA, and NNP. Each plot shows the distribution of projected points per MSE value.

imate, measured as Mean Square Error (MSE), both on training data $\mathbf{Y}_T$ and on the held-out data $\mathbf{Y}_E$. We see that, during training, both APPA and SAPPA yield MSE values (close to) zero for most points. This is evidence that using a prior to shape the learned projection – be it an explicit barrier function or a KDE directly, see Sec. 3 – allows training to focus on placing points only in regions of interest. In contrast, NNP produces such low-error projected points about half as often and also yields higher MSE values overall. On unseen data, all algorithms yield slightly larger MSE values than on training data. Yet, (S)APPA still yields lower MSE values than NNP, which numerically confirms the diffusion effect we saw.

Table 2 summarizes the MSE errors computed over all the tested dataset-projection combinations. We see that (S)APPA yield practically identical values which

are significantly better than those given by NNP.

Table 2: Mean Squared Error with respect to reference projections for APPA, SAPPA, and NNP, averaged over all studied datasets and projections. Lower is better.

	APPA	SAPPA	NNP
Train	**0.0032**	**0.0032**	0.0088
Test	0.0086	**0.0079**	0.0112

4.3 Controlling projection sharpening

As shown so far, both SAPPA and APPA substantially decrease diffusion. In particular, SAPPA does this explicitly by linearly penalizing placing points closer than a threshold δ to any barrier point $\mathbf{b}_j$ (see Sec. 3.3). In detail, the penalty term for a single projection point $\mathbf{y}_i$ and barrier point $\mathbf{b}_j$ in Eqn. 10 is

$$(\delta - \|\mathbf{y}_i - \mathbf{b}_j\|_2^2)_+ = \begin{cases} 0, \text{ if } \|\mathbf{y}_i - \mathbf{b}_j\|_2^2 \geq \delta \\ \delta - \|\mathbf{y}_i - \mathbf{b}_j\|_2^2, \text{ otherwise} \end{cases} \tag{11}$$

Hence, we can see δ as a *sharpness* parameter: The more we increase it, the farther away from the barrier points SAPPA must place its learned projection, yielding increasingly more concentrated point clusters from the training projection.

Figure 7 explores this effect. At $\delta = 0$, SAPPA reduces to NNP – there is no barrier-related penalty. We see slight sharpening with $\delta = 0.02$, with fewer points falling in the inter-cluster spaces in the projection. For $\delta = 0.04$, clusters become overall thinner or more densely packed. If we increase δ too much ($\delta = 0.1$), clusters collapse and merge, which is undesirable. In our experiments, $\delta = 0.02$ produces good sharpening results. Using δ thus allows analysts to easily control the overall sharpening effect they want to achieve in the learned projections.

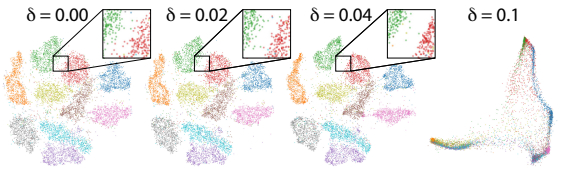

Figure 7: Setting the δ parameter in SAPPA for a t-SNE projection of the MNIST dataset. The higher δ is, the farther away from the barrier SAPPA learns to place its projection. Increasing δ sharpens the projection, up to where the projection becomes degenerate ($\delta = 0.1$).

4.4 Projecting unseen data

As already mentioned, diffusion is an artifact, we claim, of NNP *overfitting* to its training data. This matches the finding that NNP degrades strongly for data not seen during training – the network is not able to generalize well to new data. Moreover, NNP degrades further as it is asked to project increasingly more data points that were not seen during training (see Fig. 11 in (Espadoto et al., 2020b)).

One can avoid this issue if, instead of approximating P – where P is any projection algorithm – we simply re-ran P on the new data. Two main problems exist with this. First, this is highly expensive if the underlying P is itself expensive as in the case of *e.g.* t-SNE. Secondly, many such algorithms (among which t-SNE) produce *unstable* projections due to their stochastic nature. Hence, visualizing a sequence of such projections showing increasingly more data (even if drawn from the same distribution) will not allow analysts to uncover any meaningful insights.

(S)APPA strikes a good trade-off between NNP's ability to quickly produce an approximate projection for unseen data (but with high diffusion) and the exact projection P's ability to produce a projection with high quality (but in a costly and unstable manner). Figure 8 illustrates this by showing projections obtained via several methods for an increasingly large subset of the MNIST dataset. We use two different training projection techniques: t-SNE and UMAP. We then train NNP, APPA, and SAPPA to approximate each projection, using 5K training samples. We next use each of t-SNE, UMAP, as well as the three approximations of each (NNP, APPA, SAPPA) to project a growing number of unseen data points. For reference, the black-outlined images show the training projections. The results show how both t-SNE and UMAP yield well-separated clusters (in line with the known cluster separation in MNIST) but also high projection instability. At the other end of the spectrum, NNP shows very good stability – the same clusters get projected in the same areas – but *increasing* diffusion as the sample count increases. The latter is clearly a misleading artifact of NNP – we know, for this dataset, that drawing more samples does not create more mixed clusters. (S)APPA strike a good balance in the middle: Its projections show identical stability to NNP; and diffusion increases far less than NNP as the sample count increases.

4.5 Computational scalability

An approximating projection algorithm, such as (S)APPA or NNP, is only useful if it increases computing speed as compared to the reference DR algorithm it is trained on. To assess this, we compared two such DR algorithms (t-SNE and UMAP) with both NNP and (S)APPA for datasets ranging from 500 to 250K samples. In all cases, training used a maximum of 300

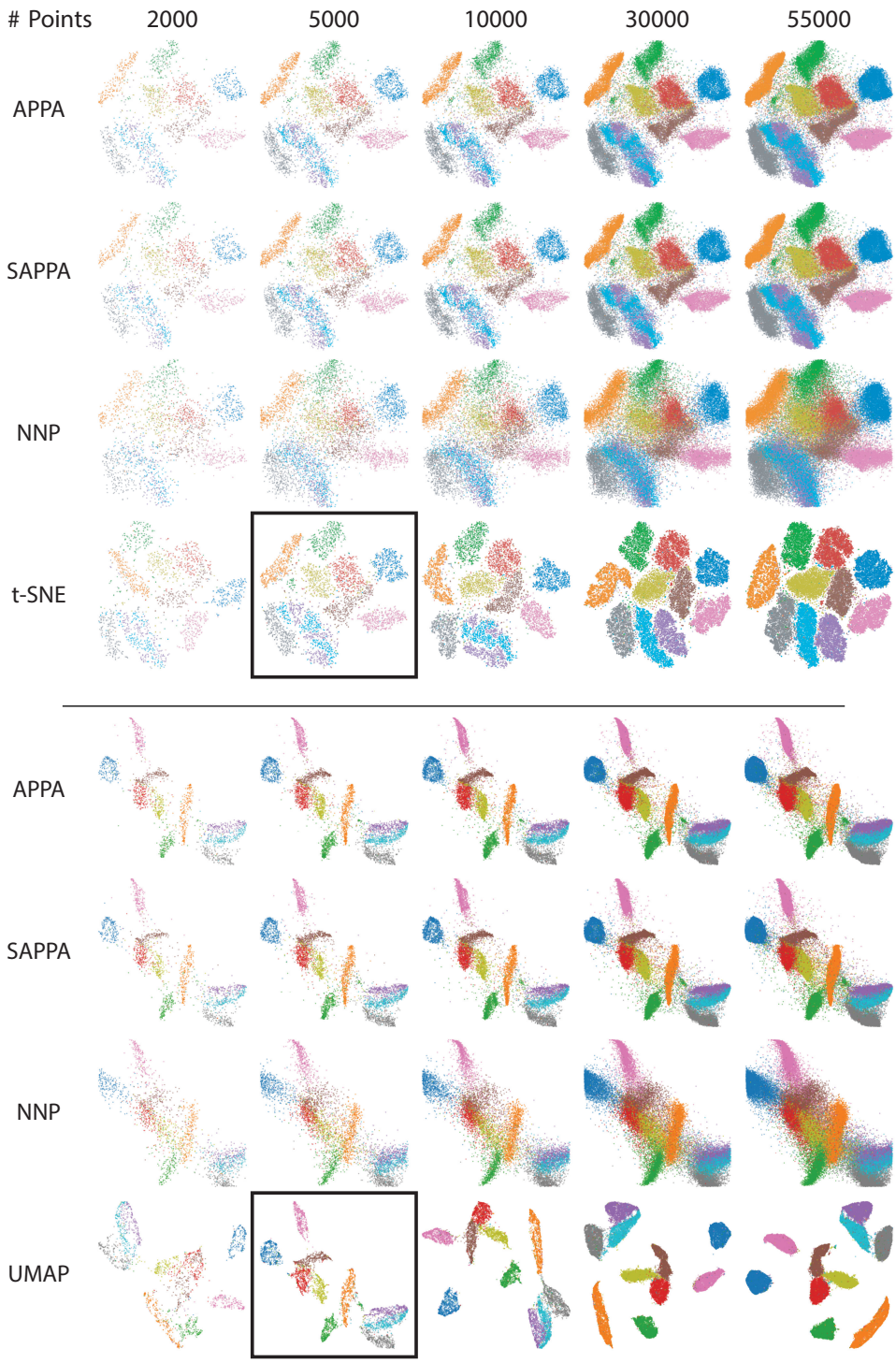

Figure 8: We train NNP and (S)APPA on a reference projection (t-SNE and UMAP, shown with a black border). We then take samples from a dataset unseen during training and compare the effects of re-running the full projection algorithm (bottom row in each group) to what we obtain by running NNP (third row) and both versions of APPA (first two rows). We see clear instability whenever we re-run the full projection algorithm and strong diffusion when we re-run NNP. (S)APPA alleviates both those issues, combining stability with significantly lower diffusion.

epochs and 5K training samples. Figure 9 shows the results. For growing dataset sizes, we see a strong increase in run time for t-SNE and UMAP. NNP runs the fastest of all due to its use of early stopping; yet, as discussed in Secs. 4.2 and 4.4, it produces lower quality, diffuse, projections. APPA's training is roughly two times slower than NNP since it does not use early stopping. SAPPA is roughly 20% slower than APPA for small datasets due to the explicit computation of distances to the barrier (Sec. 3.3). However, once trained, (S)APPA is as fast as NNP.

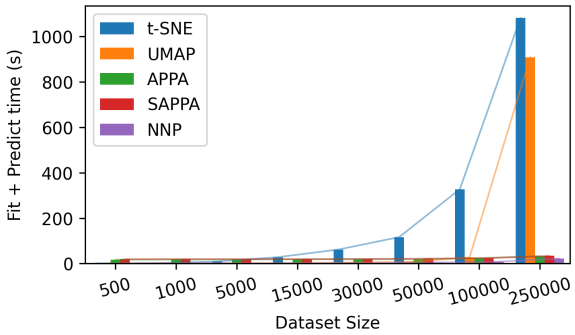

Figure 9: Computational speed of (S)APPA, NNP, t-SNE and UMAP. We train (S)APPA and NNP for at most 300 epochs with 5K samples. Each value is averaged over 3 runs.

APPA and SAPPA are both twice as slow as t-SNE for datasets of up to 5K samples. For 15K samples, (S)APPA are on average as fast as t-SNE. (S)APPA has about the same speed as UMAP for datasets with 100K samples, being slower for all smaller datasets. However, for the largest dataset size we experiment with — 250K samples — (S)APPA are between 25 and 30 times *faster* than t-SNE and UMAP. More importantly, the *trends* in this chart are telling: t-SNE and UMAP costs increase superlinearly with the sample count while both NNP and (S)APPA are linear with this count *and* have a quite low slope. All in all, this means that (S)APPA practically offer the same significant speed-ups *vs* classical projection algorithms as NNP but, as shown earlier, with increased quality.

5 DISCUSSION AND CONCLUSION

In this work, we have presented APPA and its sample-based version, SAPPA, two techniques that learn to approximate dimensionality reduction algorithms in ways that combine several desirable features. Compared to NNP, our closest competitor that we improve upon, (S)APPA cover the following points:

Genericity: As NNP, (S)APPA can learn to mimic any DR technique in a black-box fashion, that is, irrespective of the data dimensionality, nature of the data, or internals of the learned DR technique.

Quality: (S)APPA significantly improves upon NNP's weakest point, namely the latter's tendency to generate diffusion effects in projections of otherwise well-separated samples. Both visual comparisons and study of projection quality metrics show that such improvements consistently happen for all studied datasets and learned DR techniques. Quality increase is done in a principled fashion, *i.e.*, by regularizing the projection via the addition of a simple cost term that favors the creation of point distributions similar to those present in the training projection. This removes, for the first time to our knowledge, the key weakness of NNP.

Out of sample: As NNP, (S)APPA can project changing datasets in a stable manner, *i.e.*, by placing the same samples at the same locations in these multiple projections. However, while NNP shows increased diffusion as such datasets grow, (S)APPA keeps stability and only adds minimal diffusion to the process.

Ease of use: As NNP, once trained, (S)APPA requires no parameters to be set, making its use immediate. During training, SAPPA proposes one extra parameter δ that models the amount of sharpness of the resulting projections, which comes with a given preset ($\delta = 0.2$).

Scalability: As NNP, (S)APPA is very fast to train, yields good training results with about 5K training samples on all tested examples, and is linear in sample count in inference (projection) mode. While (S)APPA's training is roughly two times slower than NNP, its inference cost is identical to NNP, which is orders of magnitude smaller than expensive DR methods such as t-SNE. This makes (S)APPA an attractive method for projecting datasets of millions of samples.

Applications: As (S)APPA provides a more accurate drop-in replacement for NNP, it can be used in any applications where NNP-like approximations are relevant. For example, NNP's desirable properties have recently motivated its use in fields beyond DR such as Graph Drawing (Hartskeerl et al., 2025). (S)APPA can be directly used to improve such applications.

Limitations: (S)APPA works well to reduce diffusion if the reference projections have good cluster separation. If the reference projection places points close to each other without much inter-cluster distance, our regularization will not be as effective in reducing diffusion (see *e.g.* the PCA example in Fig. 4). Additionally, even if clusters are well-separated, they might consist of too few points. This makes the KDE p in Eqn. 6 too sharply peaked, with near-zero gradients almost everywhere, which hinders regularization effectiveness. One way to fix this – to be explored in future work – is by oversampling such small clusters according to the KDE, akin to a bootstrapping process.

We see several directions of future work. SAPPA's

training can be easily accelerated by using distance transform methods to compute the barrier, even on higher resolutions than our current 300^2 pixels, yielding finer-grained sharpening control (Eqn. 9). Adaptive diffusion can be incorporated to *e.g.* model true dataset distances in the projection while keeping clusters sharp where appropriate. Finally, (S)APPA can be easily extended to efficiently and accurately project time-dependent datasets, an area where few methods exist in the DR literature (Vernier et al., 2020).

REFERENCES

Espadoto, M., Falcão, A., Hirata, N., and Telea, A. (2020a). Improving neural network-based multidimensional projections. In *Proc. IVAPP*.

Espadoto, M., Hirata, N., and Telea, A. (2021). Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In *Proc. IVAPP*, pages 27–37. SciTePress.

Espadoto, M., Hirata, N. S. T., and Telea, A. C. (2020b). Deep learning multidimensional projections. *Information Visualization*, 19(3):247–269.

Espadoto, M., Martins, R., Kerren, A., Hirata, N., and Telea, A. (2019). Toward a quantitative survey of dimension reduction techniques. *IEEE TVCG*, 27(3):2153–2173.

Goldberger, J., Hinton, G. E., Roweis, S., and Salakhutdinov, R. R. (2004). Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press.

Hartskeerl, I., Mchedlidze, T., van Wageningen, S., Vangorp, P., and Telea, A. (2025). NNP-NET: Accelerating t-SNE Graph Drawing for Very Large Graphs by Neural Networks . In *Proc. Graph Drawing*.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *Proc. 3rd ICLR 2015*.

Kruskal, J. B. (1964a). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27.

Kruskal, J. B. (1964b). Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129.

Machado, A., Behrisch, M., and Telea, A. (2025). Extensible TensorFlow implementations of projection quality metrics. In *Proc. VisGap*. Eurographics.

Machado, A. and Telea, A. (2025a). APPA implementation. https://github.com/amreis/appa.

Machado, A. and Telea, A. (2025b). APPA supplemental materials. https://surfdrive.surf.nl/s/CjtqnZHdjmDbRea.

Machado, A., Telea, A., and Behrisch, M. (2024). Controlling the scatterplot shapes of 2D and 3D multidimensional projections. *Computers & Graphics*, 124.

Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. J. (2015). Adversarial autoencoders. *CoRR*, abs/1511.05644.

Martins, R. M., Coimbra, D., Minghim, R., and Telea, A. (2014). Visual analysis of dimensionality reduction quality for parameterized projections. *Comput. Graph.*, 41:26–42.

McInnes, L., Healy, J., and Melville, J. (2020). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426 [stat.ML].

Modrakowski, T. S., Espadoto, M., Falcão, A. X., Hirata, N. S. T., and Telea, A. (2022). Improving deep learning projections by neighborhood analysis. In *Proceedings of the 17th VISIGRAPP*, volume 1474, pages 127–152. Springer International Publishing.

Nonato, L. and Aupetit, M. (2018). Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*, 25(8):2650–2673.

Oliveira, A., Espadoto, M., Hirata, R., Hirata, N., and Telea, A. (2023). Improving self-supervised dimensionality reduction: Exploring hyperparameters and pseudo-labeling strategies. In *Communications in Computer and Information Science*, pages 135–161. Springer.

Paulovich, F. V., Nonato, L. G., Minghim, R., and Levkowitz, H. (2008). Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine and Journal of Science*, 2(11):559–572.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837.

Sips, M., Neubert, B., Lewis, J. P., and Hanrahan, P. (2009). Selecting good views of high-dimensional data using class consistency. *Computer Graphics Forum*, 28(3):831–838.

Smelser, K., Miller, J., and Kobourov, S. (2024). "Normalized Stress" is Not Normalized: How to Interpret Stress Correctly . In *Proc. IEEE BELIV*, pages 41–50. IEEE Computer Society.

Sorzano, C., Vargas, J., and Pascual-Montano, A. (2014). A survey of dimensionality reduction techniques. arXiv:1403.2877 [stat.ML].

van der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *Proc. AISTATS*, volume 5, pages 384–391.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *J Mach Learn Res*, 9(86):2579–2605.

Venna, J. and Kaski, S. (2006). Local multidimensional scaling. *Neural Networks*, 19(6):889–899.

Vernier, E., Garcia, R., da Silva, I., Comba, J., and Telea, A. (2020). Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum*, 39(3):241–252.