Improving Deep Learning Projections by Neighborhood Analysis*

Terri S. Modrakowski[1 [0000−0002−1223−8329]], Mateus Espadoto[2,3 [0000−0002−1922−4309]], Alexandre X. Falcão[4 [0000−0002−2914−5380]], Nina S. T. Hirata[2 [0000−0001−9722−5764]], and Alexandru Telea[1 [0000−0003−0750−0502]]

[1] Utrecht University, Utrecht, Netherlands
[2] University of São Paulo, Brazil
[3] University of Groningen, Groningen, Netherlands
[4] University of Campinas, Campinas, Brazil
t.s.modrakowski@students.uu.nl, {mespadot, nina}@ime.usp.br, afalcao@ic.unicamp.br, a.c.telea@uu.nl

Abstract. Visualization of multidimensional data is a difficult task, for which there are many tools. Among these tools, dimensionality reduction methods were shown to be particularly helpful to explore data visually. Techniques with good visual separation are very popular, such as those from the SNE-class, but those often are computationally expensive and non-parametric. An approach based on neural networks was recently proposed to address those shortcomings, but it introduces some fuzziness in the generated projection, which is not desired. In this paper we thoroughly explain the parameter space of this neural network approach and propose a new neighborhood-based learning paradigm, which further improves the quality of the projections learned by the neural networks, and we illustrate our approach on large real-world datasets.

Keywords: Dimensionality reduction · machine learning · neural networks · multidimensional projections.

1 Introduction

High dimensional datasets are prominent in many fields of science. However, exploring such datasets is challenging, especially when the number of dimensions – also called attributes or variables – is large. This difficulty is particularly salient for visualization methods that address high-dimensional data [16, 21, 29].

One major type of techniques used for high-dimensional visualization methods is formed by so-called *dimensionality reduction* (DR) methods. In contrast to other methods in information visualization, DR methods can handle data having a higher number of dimensions – up to hundreds or even thousands – as the visual space is not assigned separately for each dimension. Different DR techniques have been developed, aiming to balance various requirements such as speed, projection quality, and ease of use [44, 39, 29, 8]. Arguably one of the best known (and used) such techniques is t-SNE [25], which is able to create scatterplots that capture well data separation in the original space. Yet, t-SNE is slow to run on datasets of tens of thousands of

* This study was financed in part by FAPESP (2014/12236-1, 2015/22308-2 and 2017/25835-9), CNPq (303808/2018-7) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

observations or more, due to its quadratic time complexity; its hyperparameters can be hard to tune to get a good result [47]; its results are quite sensitive to data changes, *e.g.*, adding more samples may result in a completely different projection; and it cannot project out-of-sample data, which is useful for time-dependent data analysis [36, 29, 46]. Work has been done to address the performance issue, such as tree-accelerated SNE [24], H-SNE [33], A-SNE [34], and UMAP [28], which is a completely different algorithm but with the stated goal of having t-SNE quality at a higher speed. However, in general, there is no technique in the SNE class that jointly addresses scalability, stability, and out-of-sample handling.

A very different approach to projection was recently proposed [6] based on deep learning. Given a high-dimensional dataset and its 2D scatterplot created by the DR technique of choice of the user, a neural network is trained to reproduce the scatterplot. After training, the network can generate projections of high-dimensional datasets that are similar in nature to the ones used during training. This method – referred next as Neural Network Projection (NNP) – is particularly interesting as an alternative to t-SNE, since it is several orders of magnitude faster than classical t-SNE, has out-of-sample capability by design, and is simple to implement and easy to use.

A drawback of the work in [6] is, however, validation: While NNP is shown to work well, in terms of projection quality metrics, on a variety of datasets and able to learn different projection methods, the space of *hyperparameters* used during architecture and training is left unexplored. This leaves two open questions. First, a detailed study of how NNP's results depend on these hyperparameters is needed before being able to claim that the method can consistently generate good projections, as common with other deep learning evaluations [9, 13]. A second problem of NNP's results shown in [6] is that these exhibit less sharp separation of data clusters than in the ground-truth (training) projection, compare *e.g.* Figs 1a and b, a phenomenon referred to next as projection diffusion or fuzziness. Exploring how hyperparameter settings affect, and possibly reduce, diffusion is thus an important open question.

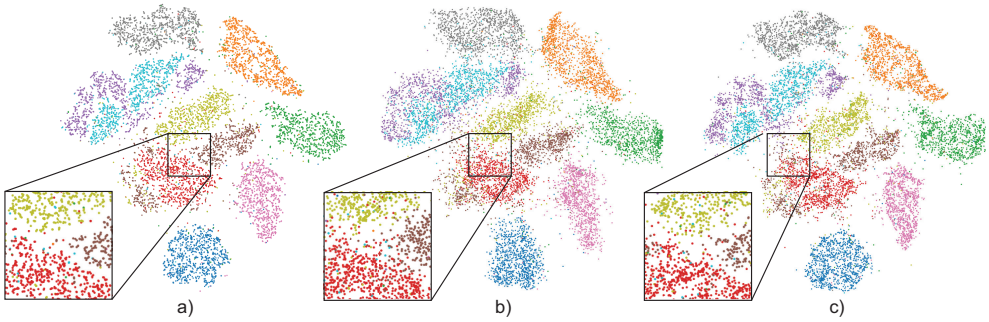


Fig. 1. Example of diffusion introduced by NNP. (a) Ground-truth t-SNE projection (b) Inferred NNP (10K samples) with diffusion. (c) Inferred KNNP (10K samples) showing less diffusion.

Many potential causes exist for diffusion, *e.g.*: (1) too small training sets or too few training epochs (underfitting); (2) using a suboptimal regularization (overfitting); (3) using an improper optimizer which gets stuck in a local minimum of the cost function. Recently, Espadoto *et al.*[7] studied the causes of diffusion by exploring

the *hyperparameter space* of NNP, showing how these influence the results' quality, gauged by projection quality metrics. They showed that NNP is *stable* with respect to hyperparameter settings, thereby completing the claim made by [6] that they can be reliably used for out-of-sample and noisy-data contexts. However, they did not propose a way to *reduce* diffusion.

In this paper, we extend the work in [7] by proposing a novel approach to deep learning projections. Rather than learning from a *single* sample at a time, we project whole groups of related (neighboring) samples at a time. This aids the network to learn how to preserve neighborhoods. To do this, we explore different schemes of efficient nearest-neighbor search in high-dimensional data during both training and inference. We evaluate our method, called K-Nearest-Neighbor Projection (KNNP) against NNP on a variety of datasets and using several quality metrics, and show that our strategy is both computationally scalable and also leads to quality improvements as compared to NNP.

The structure of this paper is as follows. Section 2 discusses related work and introduces NNP. Section 3 details our experimental NNP evaluation. Section 4 presents our results for optimizing NNP via hyperparameter space search, discussed next in Sec. 5. Section 6 presents the new KNNP projection method, whose results are discussed next in Sec. 7. Finally, Section 8 concludes the paper.

2 Related Work

2.1 Dimensionality reduction

We first introduce some notations. Let $D = \{\mathbf{x}_i\}$, $1 \leq i \leq N$ be a dataset of N samples, where each sample $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leq i \leq n$ is an n-dimensional (nD) data point. We model a projection method by a function $P : \mathbb{R}^n \to \mathbb{R}^q$ where $q \ll n$. We next consider 2D projections, *i.e.*, set $q = 2$. Hence, the projection of a sample $\mathbf{x} \in D$ is denoted by $P(\mathbf{x})$. For notation ease, we denote the 2D scatterplot obtained by projecting a dataset D as $P(D) = \{P(\mathbf{x}) | \mathbf{x} \in D\}$.

Dimensionality Reduction (DR) techniques implement various versions of the function P, aiming at optimizing different so-called quality metrics such as distance preservation or neighborhood preservation, by using different optimization strategies. Arguably the simplest, and probably earliest, DR method is PCA [32, 15], which is simple to implement, fast, and deterministic. More complex techniques include manifold learners, such as MDS [43], Isomap [42], LLE [37], and UMAP [28], which work well when the high-dimensional data is distributed over a manifold surface. These techniques are compared extensively from algorithmic viewpoints [12, 44, 5, 39, 21, 2, 50] and also from practical viewpoints (benchmarking) [8].

The t-Stochastic Neighbor Embedding projection [25] and other similar techniques in the same family [33, 24] succeed in creating high-quality projections from a neighborhood preservation perspective, thereby being very effective in exploring cases where data is segregated into similar-sample clusters in unsupervised learning contexts. However, as outlined in Sec. 1, t-SNE is quite slow and, due to its stochastic optimizer, hard to tune so that it produces deterministic results. Obviously, the latter implies it cannot be directly used in out-of-sample contexts. Parametric t-SNE variants help with this latter problem [23] at the expense of more complex, and slower, implementations.

2.2 Deep learning projections

Early on, autoencoders [11, 18] were proposed to generate a compressed, low-dimensional representation on their bottleneck layers by training the network to reproduce its inputs on its outputs. Typically, autoencoders produce results comparable to PCA. The ReNDA algorithm [1] uses two networks, improving on earlier work from the same authors. One network is used to implement a nonlinear generalization of Fisher's Linear Discriminant Analysis, using a method called GerDA; the other network is an Autoencoder used as a regularizer. The method scores well on predictability and has out-of-sample capability. However, it requires labeled data, which is not always available. Parametric t-SNE (pt-SNE) [23] was proposed to address the out-of-sample limitation of t-SNE. Being of parametric nature (mapping the entire nD input space to the lower-dimensional qD space), it allows out-of-sample behavior by construction. Only few other DR methods are parametric and thus have this ability (*e.g.*, PCA [15], NCA [10], and autoencoders [11]).

Recently, Espadoto *et al.* [6] proposed Neural Network Projections (NNP). Consider a dataset $D \subset \mathcal{D}$, where $\mathcal{D} \subset \mathbb{R}^n$ is a so-called universe of high-dimensional datasets related to a given domain, *e.g.* natural images of people faces. A training subset $D_s \subset D$ thereof is selected, and projected by some DR method (t-SNE or any other) to yield a so-called training projection $P(D_s) \subset \mathbb{R}^2$. Next, D_s is fed into a three-layer, fully-connected, regression neural network which is trained to output a 2D scatterplot $P_{nn}(D_s) \subset \mathbb{R}^2$ by minimizing the mean squared error between $P(D_s)$ and $P_{nn}(D_s)$. After that, the network is used to construct projections of unseen data from the same universe, $D_p = \mathcal{D} \setminus D_s$, by means of 2-dimensional, non-linear regression. This approach is fundamentally different from *autoencoders* [11] which do not learn from a training projection P, but simply aim to extract, in an unsupervised way, latent low-dimensional features that best represent the input data. Also, NNP is different from pt-SNE since it uses *supervised* learning (based on the training projection), has a much simpler network architecture, and a different cost function (distance to 2D ground-truth projection rather than Kullback-Leibler divergence). The NNP pipeline is shown in Figure 2.

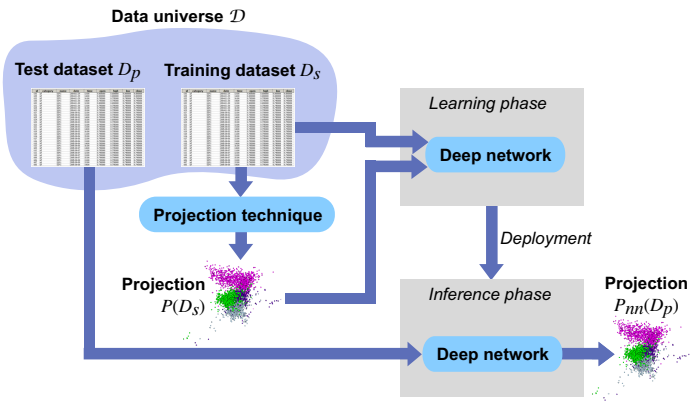


Fig. 2. Pipeline for the Neural Network Projection (NNP) method.

The NNP method is simple to implement, generically learns any projection P for any dataset $D \subset \mathbb{R}^n$, has deterministic (thus, out-of-sample) behavior, and is orders of magnitudes faster than classical projection techniques, in particular t-SNE.

However, as outlined in Sec. 1, designing a quality neural network (in our case, the NNP regressor) is challenging, given the large space of design decisions available. We outline below five such degrees of freedom, all of them relevant for NNP's performance.

Network Architecture: Much of the flexibility of neural networks (NNs) comes from architectural choices. If we follow NNP's design, which restricts itself to fully connected networks, the open choices regard the number of layers and layer sizes.

Loss functions: To start with, one needs to choose a loss function J that describes what the network aims to learn. Typical loss functions for regression are Mean Squared Error (MSE), Mean Absolute Error (MAE), logcosh, and Huber loss (see Tab. 1, where $\hat{\mathbf{y}} = \{\hat{y}_i\}$ is the NN's output vector and $\mathbf{y} = \{y_i\}$ is the training-set sample that $\hat{\mathbf{y}}$ should infer). These losses differ in several aspects: Smoother ones (MSE, logcosh) are easier to optimize by gradient descent and its variants (discussed further below); MAE is harder to optimize since its gradient is discontinuous. The Huber loss, controlled by a parameter α, behaves in the range of these models – it is close to MAE for low α and close to MSE for larger α. Finally, MAE and Huber losses are typically more robust to outliers than MSE.

Regularization: To address overfitting, regularization techniques are used to make training harder, therefore increasing the number of epochs needed for the NN to achieve convergence and thus increase their generalization power. Such techniques include L_2, L_1, max-norm, early stopping, and data augmentation. The L_k regularization techniques, also known as *weight pruning* ($k = 1$) and *weight decay* ($k = 2$), adds a penalty term $\lambda\|\boldsymbol{w}\|^k$ to the NN cost function, *i.e.*, the k-norm of the weights $\boldsymbol{w}$ of a given layer. Here, the parameter λ gives the regularization strength. The L_1 [30] variant sets the weights of less important layers to zero, thereby effectively sparsifying the model. The L_2 [19] variant, in contrast, decreases weights to small but non-zero values, thereby effectively distributing weights over more connections across the model. Both L_1 and L_2 are effective in reducing overfitting. Max-norm [40] regularization caps the norm of layer weights to a maximal value γ. Another way to reduce overfitting is to combine such regularization techniques with early stopping [51]. Hereby training is halted when the validation loss J_V stops decreasing or starts increasing while the training loss J_T keeps dropping, a situation which signals overfitting.

Optimizers: The NN cost can be minimized using Gradient Descent (GD). GD aims to minimize the training error by adjusting the weights $\boldsymbol{w}_t$ at the current step t by taking steps of size η (also called the learning rate) downwards along the gradient ∇J_t of the current value of the loss function J_t with respect to $\boldsymbol{w}$ as

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta \nabla J_t. \tag{1}$$

Applying GD (Eqn. 1) on large datasets is expensive. Stochastic Gradient Descent (SGD) speeds this up by using only one randomly-picked sample per update step t. However, this uses too little information in each update step. Mini-batch SGD blends the speed of SGD with the quality of GD by using one set (batch) of samples for each iteration of Eqn. 1. The Momentum method [35] makes SGD converge faster by

blending two consecutive updates by

$$\boldsymbol{w}^t = \boldsymbol{w}_{t-1} - \eta(\nu \nabla J_{t-1} + \nabla J_t), \tag{2}$$

where ν controls the momentum strength.

As in any optimization based on gradient descent, using a fixed learning rate η is not optimal: Small values make convergence slow (many iterations); large values may skip over minima. Adaptive Moment Estimation (ADAM) [17] improves upon this by adaptively computing η by using squared gradients. While this makes convergence faster, ADAM does not guarantee to always minimize better than fixed-learning-rate methods such as Mini-batch SGD [48].

Data augmentation: Adding data that is similar to existing training data is typically used to make deep learning effective in cases when only small training sets are available. However, data augmentation also creates models that generalize better, thereby being useful for regularization purposes.

Table 1. Typical NN loss functions. (table from [7])

Function	**Definition**
MSE	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
MAE	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$
logcosh	$\frac{1}{n}\sum_{i=1}^{n} log(cosh(y_i - \hat{y}_i))$
Huber	$\begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if} \lvert y-\hat{y}\rvert \leq \alpha \\ \alpha\lvert y-\hat{y}\rvert - \frac{1}{2}\alpha & \text{otherwise} \end{cases}$

3 NNP Evaluation

While NNP has several advantages, as mentioned in the previous section, its quality and stability *vs* hyperparameter settings has not yet been assessed in detail. We address this by performing a set of experiments that explore the design space of NNP (Sec. 3.1). For each experiment, we evaluate NNP using several quality metrics (Sec. 3.2).

3.1 Hyperparameter space exploration

To evaluate NNP's performance, we consider the space spanned by five dimensions: Network architecture, regularization methods, optimizers, data augmentation, and loss functions. We sample every dimension using several values (in terms of both method types and actual values of parameters), aiming to cover typical choices in the literature (Sec. 2) – when these are available – or choices that we deem suitable for the NNP context. All these are detailed below.

Evaluating all combinations of all sample values across all five dimensions is impractical, as it would lead to over 70 thousand of training-and-testing runs. We decrease this search space by optimizing for every dimension in turn. Early stopping was used on all experiments, stopping training if the validation loss stops decreasing for more than 10 epochs. Except when noted otherwise, we used the ADAM optimizer

and the MSE loss function. As dataset to project, we use MNIST [20], which has 70K samples of handwritten digits from 0 to 9 represented as 28x28-pixel grayscale images, flattened to 784-element vectors. We use training-sets of 2K, 5K, 10K and 20K samples picked randomly from the full dataset and a 10K test-set sample. This way, we test how *both* the hyperparameter values and the training-set size affect the quality of NNP. Note that MNIST was also used by the original NNP method [6], which makes it easy to compare our results.

We next detail the sampling of our five dimensions, pointing also at sections in the paper where the respective evaluation is detailed.

Network Architecture (Section 4.5): The NNP architecture in [6], called next *Standard*, has three fully-connected hidden layers, with 256, 512, and 256 neurons respectively. We created variants of this architecture by using a total of 360 (small), 720 (medium) and 1440 (large) neurons, distributed into three different layouts, called straight (st), wide (wd) and bottleneck (bt). This leads to exploring nine architectures. For each architecture, we list its number of neurons in the first, middle, and final layer below:

- *Small - straight*: 120, 120 and 120 neurons;
- *Small - wide*: 90, 180 and 90 neurons;
- *Small - bottleneck*: 150, 60 and 150 neurons;
- *Medium - straight*: 240, 240 and 240 neurons;
- *Medium - wide*: 180, 360 and 180 neurons;
- *Medium - bottleneck*: 300, 120 and 300 neurons;
- *Large - straight*: 480, 480 and 480 neurons;
- *Large - wide*: 360, 720 and 360 neurons;
- *Large - bottleneck*: 600, 240 and 600 neurons.

All above architectures are fully connected, and use ReLU activation, followed by a 2-unit layer which uses the sigmoid activation function to generate the 2D coordinates of the projected points.

Regularization (Section 4.1): Following Sec. 2, we studied three regularization methods:

- L_1 with $\lambda \in \{0, 0.001, 0.01, 0.1\}$ (0 denotes no regularization);
- L_2 with $\lambda \in \{0, 0.001, 0.01, 0.1\}$ (0 denotes no regularization);
- Max-norm, with $\gamma \in \{0, 1, 2, 3\}$, (0 denotes no max-norm constraint).

Optimizers (Section 4.2): We explored both ADAM and Mini-batch SGD optimizers with learning rates $\eta \in \{0.01, 0.001\}$ and momentum $\nu = 0.9$. The batch size used was 32 samples in both cases.

Data augmentation (Section 4.3): We used two variants data augmentation with the aim of reducing the diffusion effect present in the original NNP method, as follows:

- *Noise Before:* The t-SNE projection is known to create clear-separated clusters even for relatively noisy data. We aim to leverage this by artificially making the data to project more complex. For this, we add Gaussian noise of zero mean and standard deviations $\sigma \in \{0, 0.001, 0.01\}$, with 0 meaning no added noise, to the high-dimensional training data. We then project this noised data, added to the original (clean) data by t-SNE, and train NNP to mimic this projection;

– *Noise After:* The idea behind this strategy is to 'jitter' the high-dimensional data while keeping its projection fixed to that of the clean data, and train NNP to mimic the (clean) projection. This way, we hope to teach NNP that small-scale jitters in the data should not create diffusion in the projection. For this, add Gaussian noise (same σ as in the Noise Before strategy) to the data and train NNP to project the noised data dataset to obtain the clean projection.

Loss functions (Section 4.4): Following Sec. 2, we experimented with four loss functions: Mean Squared Error (MSE), used by the original NNP; Mean Absolute Error (MAE); logcosh; and Huber, with $\alpha \in \{1, 5, 10, 20, 30\}$.

3.2 Quality measurement

We measure the quality of the projections created by NNP by four metrics (see Table 2). These are well-known in the DR literature [8]. Moreover, these were also used when assessing the original NNP method.

Trustworthiness T: Gives the fraction of close points in D that are also close in $P(D)$ [45]. Differently put, a low T indicates that a projection exhibits so-called missing neighbors [27]. $U_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in 2D but not among the K nearest neighbors of point i in $\mathbb{R}^n$; and $r(i, j)$ is the rank of the 2D point j in the ordered set of nearest neighbors of i in 2D. We use $K = 7$ neighbors, following [44, 27, 8];

Continuity C: Gives the fraction of points in $P(D)$ that are also close in D [45]. A low C indicates that a projection has so-called false neighbors [27]. $V_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in $\mathbb{R}^n$ but not among the K nearest neighbors in 2D; and $\hat{r}(i, j)$ is the rank of the $\mathbb{R}^n$ point j in the ordered set of nearest neighbors of i in $\mathbb{R}^n$. As for T, we use $K = 7$ neighbors;

Neighborhood Hit NH: For labeled data, NH tells how homogeneous point-clusters are in the projection, ranging from perfect separation ($NH = 1$) to no separation ($NH = 0$) [31]. For data which one knows that consists of well-separated sample clusters, one would want a high NH projection. NH is the number $\mathbf{y}_K^l$ of the K nearest neighbors of a point $\mathbf{y} \in P(D)$, denoted by $\mathbf{y}_K$, that have the same label as $\mathbf{y}$, averaged over $P(D)$. As above, we use $K = 7$ neighbors;

Shepard diagram correlation R: Shepard diagrams are scatterplots of Euclidean distances (in nD *vs* 2D) between all point-pairs [14]. Diagrams that are close to the main diagonal indicate that the projection preserves distances well. Following [8], we measure the quality implied by a Shepard diagram by computing the Spearman ρ rank correlation of the respective 2D scatterplot, with $R = 1$ telling a perfect distance correlation.

4 NNP Evaluation Results

We next present the results of the NNP hyperparameter-space exploration performed along the five dimensions described in Sec. 3.

Table 2. Quality metrics. Right column gives metric ranges, with optimal values in bold.

Metric	Definition	Range
T	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in U_i^{(K)}} (r(i,j) - K)$	$[0, \mathbf{1}]$
C	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in V_i^{(K)}} (\hat{r}(i,j) - K)$	$[0, \mathbf{1}]$
NH	$\frac{1}{N} \sum_{\mathbf{y} \in P(D)} \frac{\mathbf{y}_k^l}{\mathbf{y}_k}$	$[0, \mathbf{1}]$
R	$\rho(\|\mathbf{x}_i - \mathbf{x}_j\|, \|P(\mathbf{x}_i) - P(\mathbf{x}_j)\|), 1 \leq i \leq N, i \neq j$	$[0, \mathbf{1}]$

Table 3. Effect of **regularization**. Rows show metrics for t-SNE (GT row) *vs* NN projections using different training-set sizes. Bold shows values closest to GT. (table from [7])

a) L_1 **regularization**

Model	λ	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	**0.705**	**0.843**	**0.957**	**0.443**	50	6.20
	0.001	0.677	0.827	0.948	0.439	58	7.14
	0.01	0.660	0.815	0.945	0.438	94	10.93
	0.1	0.632	0.806	0.943	0.454	82	9.98
5K	0	**0.738**	**0.871**	**0.962**	**0.423**	26	7.22
	0.001	0.692	0.845	0.953	0.436	38	10.05
	0.01	0.670	0.835	0.947	0.427	68	18.29
	0.1	0.599	0.815	0.945	0.459	53	14.58
10K	0	**0.834**	**0.902**	**0.968**	**0.337**	45	22.32
	0.001	0.753	0.852	0.958	0.348	31	16.09
	0.01	0.722	0.833	0.951	0.352	39	19.12
	0.1	0.665	0.811	0.947	0.345	61	30.67
20K	0	**0.885**	**0.922**	**0.967**	**0.341**	49	47.28
	0.001	0.816	0.883	0.960	0.364	30	29.35
	0.01	0.743	0.842	0.954	0.366	28	26.89
	0.1	0.707	0.822	0.946	0.364	25	24.17

b) L_2 **regularization**

NH	T	C	R	# epochs	Time (s)
0.929	0.990	0.976	0.277		
0.695	0.839	0.956	0.437	35	4.61
0.711	**0.847**	**0.958**	0.432	29	4.27
0.684	0.834	0.954	0.433	42	5.57
0.683	0.830	0.952	**0.428**	68	8.54
0.767	**0.880**	**0.963**	0.422	53	14.33
0.742	0.875	**0.963**	0.419	28	7.71
0.733	0.866	0.959	**0.416**	55	15.24
0.709	0.860	0.958	0.429	45	12.51
0.833	**0.899**	**0.967**	0.342	43	20.93
0.821	**0.899**	0.966	0.340	55	27.88
0.798	0.880	0.963	0.337	34	17.37
0.773	0.865	0.961	**0.336**	36	18.48
0.885	**0.922**	**0.967**	0.341	46	43.49
0.870	0.915	0.966	0.343	34	33.06
0.853	0.902	0.963	0.344	40	38.05
0.826	0.883	0.960	**0.339**	38	37.87

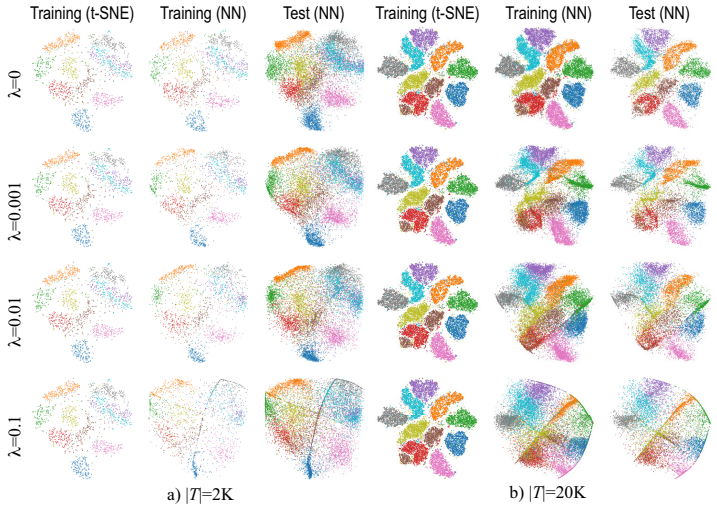


Fig. 3. L_1 **regularization:** Effect of λ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set. (figure from [7])

4.1 Regularization

We first focus on studying the L_1 and L_2 regularization methods. For both of them, used independently, we use increasing amounts (controlled by λ), aiming to see whether higher regularization, which makes learning harder, can improve the quality of the inferred projections.

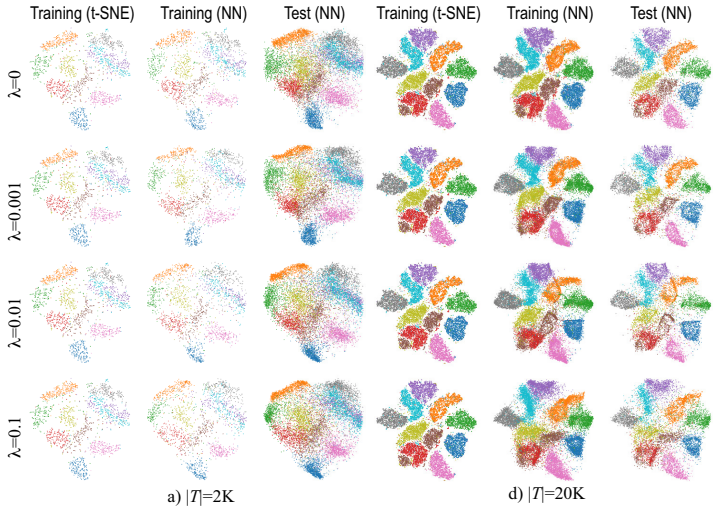


Fig. 4. L_2 **regularization:** Effect of λ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set. (figure from [7])

Figures 3 and 4 show the resulting projections for the training set and test set. Given the limited space, we only show here the projections for the 2K and 20K training-set sizes – the ones for 5K and 10K training-set sizes are very similar. Here and next, 'Training (t-SNE)' shows the ground-truth (GT) projection computed by t-SNE on the training set, which is the projection whose quality we want to ultimately achieve. We see that, as λ increases, the results get worse (fuzzier, farther from the crisp separation visible in the GT projection). Separately, we see that L_2 regularization produces projections which are closer to GT than those produced using L_1 for the same λ values. Table 3 confirms these visual insights by showing the values of the four quality metrics (Sec. 3.2) for the L_1 and L_2 experiments, for all four training-set sizes (leftmost column). As noted above, L_2 regularization yields NH, T, and C values closer to the GT ones than L_1 regularization. Separately, Table 3 shows that NNP yields *higher* Shepard correlation R values than GT, for all λ values (slightly higher for L_1 than L_2). In other words, NNP preserves the nD distances in the 2D projection *better* than t-SNE projection (see definition of R, Tab. 2). This indirectly explains the diffusion we see in NNP: Indeed, since NNP aims to preserve distances, it cannot rearrange points as freely as t-SNE does (which only aims to preserve neighborhoods), thereby yielding a less crisp cluster separation than t-SNE.

The rightmost two columns in Tables 3(a,b) show the *training effort* until convergence (number epochs and seconds). Convergence is reached in under 70 epochs, regardless of the regularization type (L_1 or L_2) or strength λ. L_1 and L_2 regularization require similar effort, with L_2 being slightly faster than L_1 for smaller training sets. This confirms the initial findings from [6] that NNP converges well, and additionally tells that this happens regardless of regularization.

We next study the max-norm regularization. Figure 5 shows that the projection quality does not strongly depend on the max-norm constraint γ. The metrics in Tab. 4(a) confirm this. We also see that max-norm yields higher projection quality values (closer to GT) than L_1 and L_2. Effort-wise, max-norm regularization is very sim-

ilar to L_1 and L_2 (compare rightmost columns in Tab. 4 (a) with those in Tab. 3(a,b)). Putting all above results together, we conclude that regularization does not bring a significant benefit to NNP, with max-norm being only slightly better than L_1 and L_2.

Table 4. Effect of **max-norm** (a) and **optimizers** (b). Metrics shown for t-SNE (GT row) *vs* NN projections using different training-set sizes. Bold shows values closest to GT. (table from [7])

a) **Max-norm regularization**

Model	γ	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	**0.701**	0.839	0.956	0.443	44	5.45
	1	0.692	0.836	0.956	**0.431**	32	4.47
	2	0.699	**0.842**	**0.957**	0.441	45	5.80
	3	0.698	0.837	0.956	0.441	31	4.47
5K	0	**0.759**	**0.881**	**0.964**	**0.417**	51	13.34
	1	0.756	0.880	**0.964**	0.421	40	10.70
	2	0.740	0.866	0.961	0.420	24	7.05
	3	0.755	0.879	0.963	0.423	48	13.23
10K	0	0.824	0.898	0.966	**0.337**	37	18.14
	1	**0.840**	0.904	0.967	0.338	31	15.43
	2	0.829	0.903	0.967	0.340	37	18.67
	3	0.837	**0.905**	**0.968**	0.338	53	26.63
20K	0	**0.886**	**0.923**	**0.967**	0.342	56	52.65
	1	0.870	0.918	0.967	**0.340**	26	25.22
	2	0.881	0.917	0.967	0.341	30	28.88
	3	0.879	0.920	0.967	0.345	34	34.11

b) **Optimizers**

Model	Optimizer (η)	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	ADAM	**0.696**	**0.841**	**0.956**	**0.447**	30	3.72
	SGD (0.01)	0.625	0.791	0.938	0.464	97	8.32
	SGD (0.001)	0.610	0.787	0.938	0.464	455	36.56
5K	ADAM	**0.733**	**0.861**	**0.960**	**0.421**	19	5.27
	SGD (0.01)	0.655	0.817	0.945	0.439	86	16.90
	SGD (0.001)	0.641	0.808	0.942	0.443	402	77.17
10K	ADAM	**0.842**	**0.905**	**0.968**	**0.343**	56	26.51
	SGD (0.01)	0.707	0.821	0.949	0.362	75	28.55
	SGD (0.001)	0.690	0.812	0.948	0.360	392	147.60
20K	ADAM	**0.882**	**0.920**	**0.968**	**0.339**	43	40.77
	SGD (0.01)	0.769	0.838	0.952	0.356	129	94.30
	SGD (0.001)	0.754	0.836	0.952	0.370	423	309.19

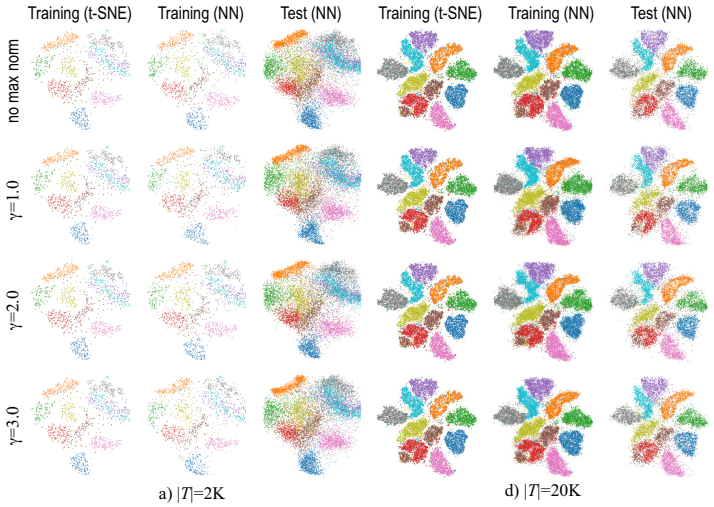


Fig. 5. Max-norm: Effect of γ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set. (figure from [7])

4.2 Optimizer

Following Sec. 3, we trained NNP using the ADAM optimizer with its default settings, and also with Mini-batch SGD with learning rates $\eta \in \{0.01, 0.001\}$. Figure 6 shows

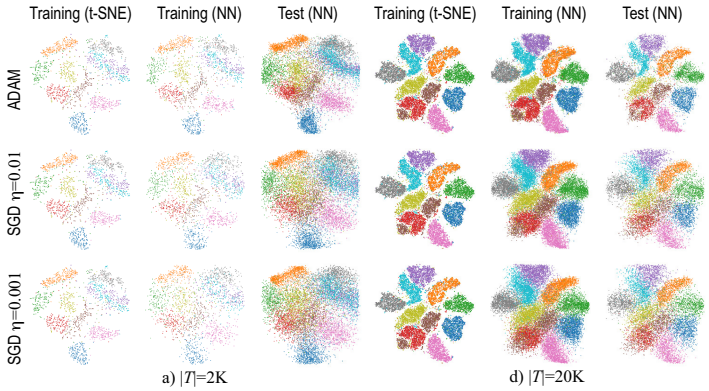


Fig. 6. Optimizer: Effects of different settings (ADAM, SGD with $\eta \in \{0.001, 0.01\}$) for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set. (figure from [7])

that ADAM produces results with far less diffusion than SGD. Table 4(b) confirms this, as ADAM scores better than SGD for all four quality metrics. We also see here that ADAM converges much faster than SGD. Since, additionally, ADAM works well with its default parameters, we conclude that this is the optimizer of choice for NNP.

4.3 Data augmentation

As outlined in Sec. 3.1, we use two data augmentation strategies to add noise in the attempt of forcing NNP to learn to reduce diffusion in the projections it creates. Figure 7 shows that the *Noise before* and *Noise after* strategies yield similar results, which are also close to GT. Table 5(a,b) confirms this, and shows that the *Noise after* strategy yields slightly higher quality metrics on average than *Noise before*. Comparing these quality values with those obtained by varying regularization techniques and optimizers (Tables 3-4), we see that *Noise after* slightly improves the projection quality. This can be explained by the 'jitter' effect of Noise after, which effectively teaches NNP that points which are at slightly different locations (in nD) should project to the *same* location (in 2D, given by the GT projection).

Table 5. Effect of **data augmentation**. Rows show metrics for t-SNE (GT row)) *vs* NNP (other rows). Right two columns in each table show training effort (epochs and time). Bold shows values closest to GT. (table from [7])

a) **Noise after** strategy

Model	Noise σ	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	0.717	0.846	0.958	0.448	31	8.84
	0.001	0.726	0.852	**0.960**	**0.430**	37	10.57
	0.01	**0.729**	**0.856**	**0.960**	0.433	54	14.52
5K	0	**0.783**	0.892	**0.966**	**0.401**	43	23.28
	0.001	0.780	**0.895**	**0.966**	0.408	52	29.44
	0.01	**0.783**	0.892	**0.966**	**0.401**	47	26.84
10K	0	**0.849**	0.909	**0.968**	0.339	44	44.06
	0.001	0.844	0.909	**0.968**	0.337	36	37.76
	0.01	0.848	**0.910**	**0.968**	**0.333**	59	60.31
20K	0	0.887	0.924	0.966	0.340	55	105.87
	0.001	**0.888**	0.924	**0.967**	**0.336**	46	88.09
	0.01	0.885	**0.925**	**0.967**	0.339	51	97.06

b) **Noise before** strategy

NH	T	C	R	# epochs	Time (s)
0.929	0.990	0.976	0.277		
0.712	**0.842**	0.957	0.446	23	7.17
0.679	**0.842**	**0.959**	0.422	33	9.57
0.682	0.833	0.956	**0.421**	20	6.86
0.785	**0.894**	0.966	0.401	62	33.34
0.793	0.884	0.966	**0.364**	31	17.91
0.802	0.888	**0.967**	0.366	49	28.68
0.849	**0.908**	**0.968**	0.336	36	35.61
0.798	0.901	0.966	0.304	37	39.58
0.802	0.904	0.966	**0.302**	53	55.58
0.888	**0.925**	**0.967**	**0.337**	40	76.08
0.865	0.920	**0.967**	0.385	42	81.55
0.869	0.920	**0.967**	0.392	41	80.52

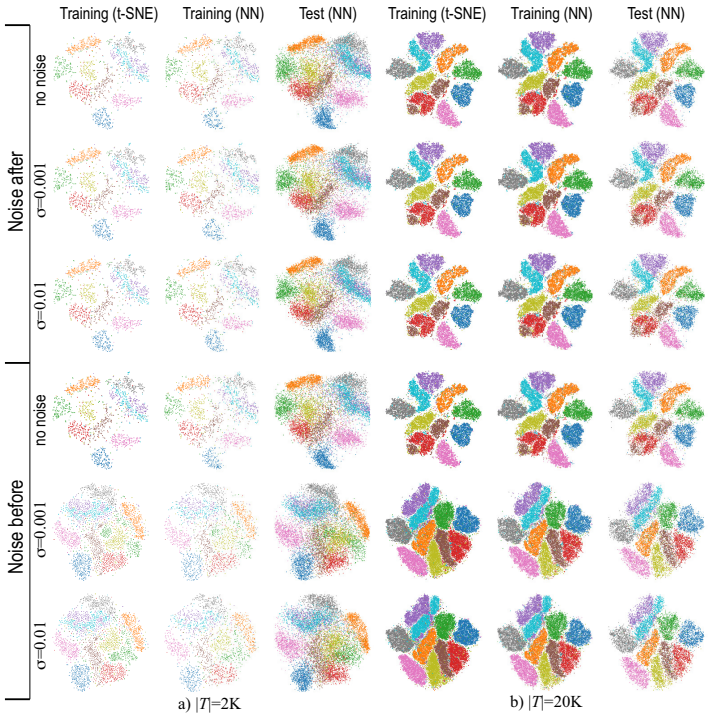


Fig. 7. Noise after and **noise before** data augmentation: Effect of noise strength $\sigma \in \{0, 0.001, 0.01\}$. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set. (figure from [7])

4.4 Loss function

Figure 8 shows the comparative effect of the four tested loss functions. Among these, MAE creates slightly less diffusion than the other ones, especially for small training sets. Table 6 (a) confirms this: MAE yields an increase of NH from roughly 0.70 (when using the other loss functions) to roughly 0.74 for the smallest test-set of 2K samples; for the largest test-set of 20K samples, the NH increases from roughly 0.87 to 0.88. Also, MAE yields the best quality metrics for all tested configurations. However, Table 6 (a) also shows that the training effort for MAE is higher than for the other loss functions. As the training set increases, the training-effort difference between MAE and the other loss functions decreases. Hence, for a real-world configuration, MAE is not really more costly than the alternatives. Given all above, we conclude that MAE is the best loss function for NNP.

4.5 Network Architecture

Our final evaluation considers using different NN architectures. Figure 9 shows that quality increases with the architecture size. This is not too surprising, since larger architectures allow more freedom to learn the desired projection patterns, and in the same time they are not too large to require more training data. Separately, we see that the *Large - bottleneck* architecture produces visual clusters that are slightly

Table 6. Effect of different **loss functions** (a) and **architectures** (b). Rows show metrics for t-SNE (GT row) *vs* NN projections using different training-set sizes. Bold shows values closest to GT. (table from [7])

a) **Loss Functions**

Model	Loss (α)	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	Huber (1.0)	0.706	0.839	0.956	0.445	34	5.94
	Huber (5.0)	0.687	0.827	0.953	0.447	16	3.99
	Huber (10.0)	0.704	0.839	0.957	0.431	45	7.53
	Huber (20.0)	0.692	0.835	0.956	0.442	32	5.88
	Huber (30.0)	0.695	0.836	0.956	0.433	30	5.98
	logcosh	0.704	0.839	0.957	0.434	33	6.37
	MAE	**0.742**	**0.866**	**0.962**	**0.423**	78	11.05
	MSE	0.704	0.842	0.957	0.442	40	6.86
5K	Huber (1.0)	0.762	0.883	0.964	0.420	50	14.50
	Huber (5.0)	0.745	0.871	0.963	0.426	26	8.34
	Huber (10.0)	0.769	0.886	**0.965**	**0.416**	69	19.79
	Huber (20.0)	0.763	0.884	**0.965**	0.420	62	18.18
	Huber (30.0)	0.768	0.883	**0.965**	0.420	55	16.16
	logcosh	0.768	0.883	**0.965**	0.425	54	16.03
	MAE	**0.781**	**0.893**	**0.965**	0.418	57	16.56
	MSE	0.753	0.874	0.963	0.428	30	9.67
10K	Huber (1.0)	0.831	0.898	0.968	0.338	38	19.56
	Huber (5.0)	0.833	0.902	0.968	0.342	40	20.64
	Huber (10.0)	0.837	0.906	**0.969**	0.344	52	26.98
	Huber (20.0)	0.831	0.900	0.968	0.344	38	19.97
	Huber (30.0)	0.831	0.902	0.968	0.348	36	19.55
	logcosh	0.818	0.893	0.967	0.347	25	14.00
	MAE	**0.848**	**0.912**	0.968	**0.333**	58	29.55
	MSE	0.839	0.906	0.968	0.339	65	32.42
20K	Huber (1.0)	0.856	0.907	0.967	0.353	20	19.57
	Huber (5.0)	0.881	0.918	0.967	0.344	44	41.69
	Huber (10.0)	0.882	0.921	**0.968**	0.344	36	34.48
	Huber (20.0)	0.881	0.920	0.967	0.342	45	43.96
	Huber (30.0)	0.877	0.915	0.967	0.341	29	28.71
	logcosh	0.884	0.919	0.967	**0.335**	35	35.46
	MAE	**0.887**	**0.927**	0.966	0.339	47	44.55
	MSE	0.871	0.914	0.967	0.341	23	21.68

b) **Network Architecture**

Model	NN Arch	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277	0	0
2K	small st	0.680	0.827	0.951	0.437	30	5.11
	small bt	0.670	0.819	0.950	0.453	18	3.85
	small wd	0.672	0.820	0.949	0.463	17	3.82
	medium st	0.683	0.827	0.952	0.441	17	3.83
	medium bt	0.690	0.833	0.955	0.456	25	4.96
	medium wd	0.702	0.838	0.956	0.438	44	7.17
	large st	0.692	0.835	0.956	0.447	19	4.66
	large bt	**0.720**	**0.852**	**0.961**	**0.430**	50	8.95
	large wd	0.713	0.847	0.959	0.434	45	8.30
5K	small st	0.744	0.875	0.962	0.414	66	18.31
	small bt	0.719	0.855	0.958	0.423	17	5.78
	small wd	0.726	0.864	0.959	0.424	40	12.21
	medium st	**0.761**	0.879	0.963	0.418	42	12.67
	medium bt	0.742	0.872	0.962	0.426	33	10.54
	medium wd	0.740	0.873	0.963	0.419	40	12.53
	large st	0.752	0.874	**0.964**	**0.408**	29	10.95
	large bt	**0.761**	**0.880**	**0.964**	0.420	34	12.02
	large wd	0.755	0.878	**0.964**	0.423	38	13.87
10K	small st	0.818	0.893	0.966	0.338	43	21.02
	small bt	0.820	0.893	0.966	0.330	54	27.35
	small wd	0.794	0.879	0.963	0.330	28	15.35
	medium st	0.828	0.900	0.968	0.343	45	22.74
	medium bt	0.820	0.895	0.967	0.343	31	16.68
	medium wd	0.825	0.899	0.967	**0.338**	49	25.74
	large st	0.831	0.902	0.968	**0.338**	32	20.34
	large bt	**0.836**	**0.905**	**0.969**	0.341	36	21.97
	large wd	0.830	0.900	0.968	**0.338**	30	19.31
20K	small st	0.865	0.910	0.965	**0.335**	30	30.81
	small bt	0.838	0.891	0.965	0.353	16	15.74
	small wd	0.865	0.910	0.965	0.345	37	34.66
	medium st	0.882	0.922	**0.967**	0.339	45	41.97
	medium bt	0.882	0.921	**0.967**	0.340	45	41.35
	medium wd	0.874	0.917	**0.967**	0.346	34	32.92
	large st	0.886	0.924	**0.967**	0.340	45	50.74
	large bt	**0.890**	**0.925**	**0.967**	0.342	37	42.48
	large wd	0.878	0.917	**0.967**	0.345	29	33.03

sharper than the ones created by the other eight studied architectures. The quality metrics in Table 6 (b) confirm this: *Large - bottleneck* has a NH about 0.04 higher for all training-set sizes. Separately, we see that, while this architecture is larger than the others, its training effort is quite similar to the others. Hence, we choose this architecture as our best one for NNP.

It is interesting to consider why the bottleneck architecture performs better than the others. At a high level, this architecture is reminiscent of the bottleneck structure used by autoencoder (AE) networks, which, as discussed in Sec. 2, are also used for dimensionality reduction. It is possible that NNP's bottleneck layer acts in a conceptually similar way to the AE one, that is, extracts latent features from the data, thereby helping the final layer(s) to create the 2D projection. However, NNP and AE are fundamentally different: While AE is driven *purely* by latent feature extraction, NNP is driven purely by the aim of mimicking a given 2D projection, constructed by completely different mechanisms (*e.g.*, t-SNE). Studying how bottleneck architectures could further improve NNP is a potential future work direction.

5 Insights from Evaluation

We next summarize the insights from the evaluation of NNP along the five dimensions discussed in Sec. 3.

Optimal settings: We obtain the best quality (that is, closest to the ground-truth t-SNE projection following our four considered quality metrics) with no regularization, ADAM optimizer, *Noise after* data augmentation, MAE loss function, and the *Large - Bottleneck* architecture. The only free parameter in this configuration is σ – the noise

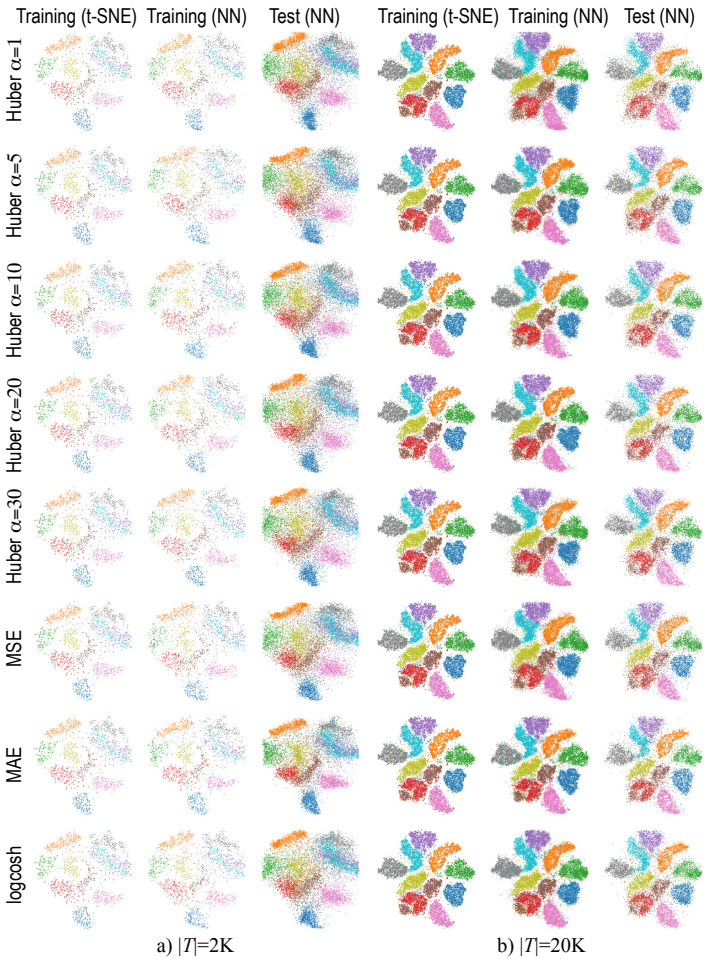


Fig. 8. Loss: Effect of different loss functions. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set. (figure from [7])

standard deviation for the data augmentation step. As Tab. 5a shows, σ affects the projection quality only very little, so we can in practice fix this parameter. We do this to a default of $\sigma = 0.01$.

As explained at the beginning of Sec. 3, we evaluated the five dimensions of NNP's hyperparameter choices *independently*, to limit the number of tested combinations, and found the above mentioned optimal settings. It is, at this point, important to test that their *combination* still yields good results. For this, we did a final experiment as follows. We used both NNP's original *Standard* architecture and the *Large - Bottleneck* architecture, both using the optimal settings, to better assess the effect of the architecture change. This is motivated by the fact that the architecture change is the largest deviation from the original NNP parameter values presented in [6]. Table 7 shows that *Large - Bottleneck* performs better than *Standard* on practically all metrics and for all training-set sizes. This improvement can be seen even when compared to the best results of each individual test, especially for smaller training-set sizes. This

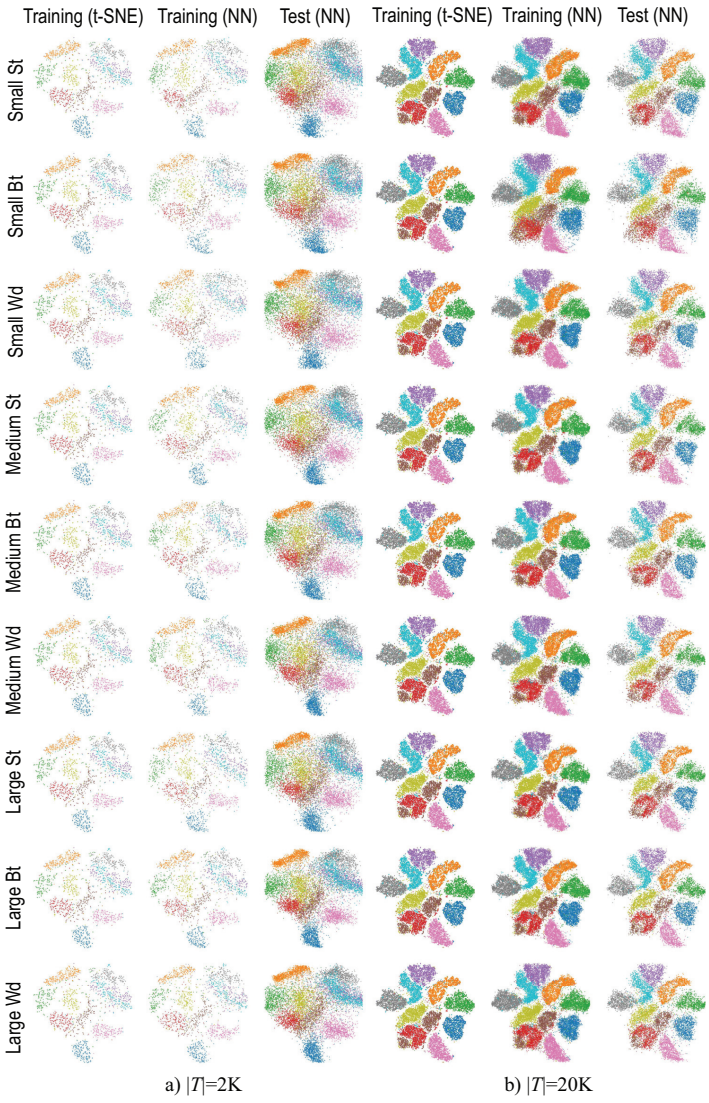


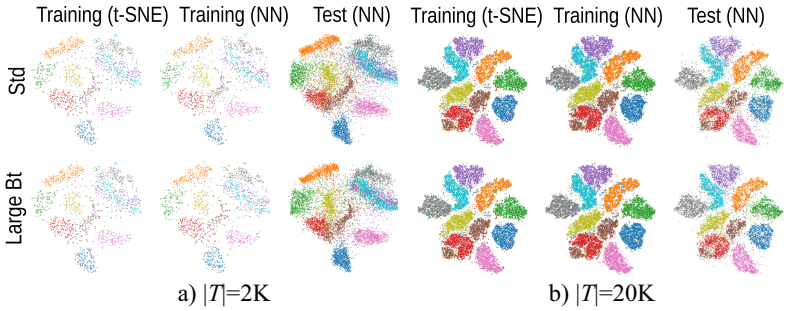
Fig. 9. Arch: Effect of different architectures. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set. (figure from [7])

improvement is also visible in the figure below Table 7 in the form of less fuzziness and better separated clusters in the images produced with *Large - Bottleneck* (bottom row) as compared to the ones produced by *Standard* (top row).

Quality: We gauged the quality of NNP by measuring four projection quality metrics: neighborhood hit, trustworthiness, continuity, and Shepard diagram correlation. The optimal hyperparameter setting presented in the beginning of Sec. 5 yields quality values that are *closer* to the ground-truth (t-SNE) values than the results presented by the original NNP in [6]. Separately, we see that, as the training-set increases (from 2K to 20K samples), the NNP quality *consistently* approaches the ground-truth

Table 7. Effect of using **optimal settings**. Top: Metrics shown for t-SNE (GT row) *vs* NN projections using different training-set sizes. Bold shows values closest to GT. Bottom: Optimal settings for *Std* and *Large - Bottleneck* NN architectures. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set. (table from [7])

Model	NN Arch	NH	T	C	R	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277	0	0
2K	std	0.753	0.871	0.963	0.433	73	14.58
	large bt	**0.773**	**0.878**	**0.964**	**0.426**	82	18.44
5K	std	0.794	0.904	0.964	**0.411**	129	60.26
	large bt	**0.813**	**0.906**	**0.966**	**0.411**	70	37.19
10K	std	**0.850**	**0.916**	**0.967**	0.334	113	104.39
	large bt	**0.850**	0.913	0.966	**0.331**	108	112.53
20K	std	0.884	0.923	**0.964**	**0.335**	121	215.66
	large bt	**0.891**	**0.929**	**0.964**	**0.335**	101	205.07



quality – see Tables 3-6. NNP quality is lower than GT quality, but the difference is under 5% on average for the 20K point training-set. This difference, however small, is still visible in the projections – visual examination of NNP shows still present diffusion as compared to the GT (t-SNE). While this diffusion *decreases* with training-set size, it is still there even for the optimal parameter settings and 20K training samples – compare *e.g.* the inference on unseen data in Fig. 7(b), Test (NN), with Fig. 7(b), Training (t-SNE). To decrease this diffusion further, something more drastic than changing hyperparameters is needed. We present such an alternative next in Sec. 6.

Stability: Our experiments show that NNP is stable with respect to training set sizes and all studied hyperparameter variations. Figures 3-9 show that NNP creates practically clusters with the same *shape* and relative *positions* in the test projections (NNP run on unseen data) as the ground-truth t-SNE projections create, for all tested configurations – that is, excluding the diffusion effect already discussed above. The stability of NNP with respect to changes in the training data, hyperparameter settings, and noise added to data is in *stark contrast* with the instability of the t-SNE projection technique with respect to *all* these three factors, and is explained by the deterministic nature of NNP's underlying neural network. This stability is of important practical value in many applications that require predictability and repeatability when creating a projection from data [47].

6 Improving NNP by Neighborhood Analysis

Following our analysis of the NNP evaluation (Sec. 5, we see that NNP scores very well on stability and quality consistency with respect to hyperparameter values. In

the same time, the quality is still on average 5% lower than that of the ground-truth (t-SNE) projection. This is visible in the still higher diffusion of NNP as compared to t-SNE. Our experiments show that hyperparameter settings, including regularization, data augmentation, optimizer, loss function and network architecture cannot fully eliminate diffusion, although by using MAE as loss function, quality metrics increased in value.

The strong visual separation of data clusters produced by t-SNE is likely one of the most praised feature of this method. t-SNE achieves this by essentially considering the preservation of *neighborhoods* rather than of point-pair distances. We next leverage this intuition in the context of NNP's deep learning approach to projections.

Consider the NNP approach, where each training sample $\mathbf{x}$ is fed into the network with its corresponding ground-truth (t-SNE) coordinate $P(\mathbf{x})$ as a training label. We replace each such training pair $(\mathbf{x}, P(\mathbf{x}))$ with a pair of *neighborhoods* $(\nu(\mathbf{x}), P(\nu(\mathbf{x}))$. Here, $\nu(\mathbf{x})$ are the K nearest neighbors of $\mathbf{x}$ in D; and $P(\nu(x))$ are the ground-truth projections of these neighbors. We compute neighborhoods ν using both a fast approximate nearest-neighbor search [26] and an exact, slower, brute-force search, to check whether the approximate search has any negative impact on quality. We call our new model K-nearest-neighbors NNP, or KNNP.

During inference, we compute nearest neighbors over points from the training set. There are two reasons for this: (1) The training set is already learned (known) by the network; (2) The training set is already indexed for fast search [26].

We tune the hyperparameters of the KNNP model following the results in Sec. 5. We use MAE as our loss function, which is averaged over the K neighbors as each one is treated as a single sample or label. We chose ADAM as our optimizer. The architecture of the network follows the one in Sec. 3.1 aside from the input and output layers which are scaled so that each input layer containing K nD points outputs a single 2D point.

7 KNNP Evaluation

We next compare the KNNP method introduced in Sec. 6 with the original NNP method using the optimized hyperparameter settings from Sec. 4 and with the ground-truth t-SNE projection. For this, we use the four quality metrics in Sec. 3.2. In addition to the MNIST dataset (Sec. 3.1), we use three more datasets to the comparison, namely:

Fashion MNIST [49]: 70K observations of 10 types of pieces of clothing, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors;

Dogs *vs* Cats [4]: 25K images of varying sizes divided into two classes (cats, dogs). We used the Inception V3 [41] Convolutional Neural Network (CNN) pre-trained on the ImageNet data set [3] to extract features of those images, yielding 2048-element vectors for each image;

IMDB Movie Review [22]: 25K movie reviews from which 700 features were extracted using TF-IDF [38], a standard method in text processing.

We next show the performance of KNNP *vs* NNP and t-SNE, for training data (Sec. 7.1) and test data (Sec. 7.2). We also show how quality depends on the training

set size (Sec. 7.3) and evaluate KNNP's speed *vs* other techniques (Sec. 7.4). Finally, we show actual projection plots computed by KNNP, NNP, and t-SNE (Sec. 7.5). Due to space restrictions, we present only a subset of our results.

7.1 Quality on training data

Figure 10 compares the performance of KNNP, the original method (NNP), and the ground truth (GT, t-SNE) across four quality metrics. Red, yellow, and green indicate that the respective method has a quality lower than, similar to, respectively higher than GT. We see that, for $K = 5$ neighbors, KNNP performs slightly better than NNP, in virtually all cases and for all quality metrics. We also see that quality does not vary much with architecture style or size. Hence, when running on a tight computational budget (where one cannot train or test large architectures), KNNP has a small edge over NNP.

7.2 Quality on testing data

So far, we compared both deep learning projections (KNNP and NNP) against each other and against the GT (t-SNE). For testing data, we cannot do the latter comparison, since t-SNE is not a deterministic method, and does not have an out-of-sample capability. Hence, for testing data, we next compare KNNP and NNP – trained on the same data, and tested on the same data – against each other only.

Figure 11 shows that KNNP gets the largest quality boost *vs* NNP for $K = 5$ neighbors again. As in Fig. 10, the style and size of architecture do not influence the results. Overall, KNNP yields better quality than NNP. However, which metric (of the four evaluated) is most improved depends on the dataset. This is expected, since neither NNP nor KNNP do *explicitly* optimize for a given quality metric.

7.3 Quality as function of training set size

Figure 12 shows how the quality of KNNP compares to that of NNP for different training-set sizes. We see that the added-value of KNNP *vs* NNP is higher for fewer training samples, particularly so for $K = 5$ neighbors. Hence, when the user can only use a small training-set, the relative added-value of KNNP *vs* NNP increases.

7.4 Computational scalability

We next compare the speed of KNNP, NNP, and other well-known techniques for up to 1M test samples. All methods were run on a 4-core Intel E3-1240v6 at 3.7 GHz with 64 GB RAM and an NVidia GeForce GTX 1080 Ti GPU with 11 GB VRAM. Figure 13a shows the projection time (log scale) as a function of the dataset size for *parametric* techniques. We see that all techniques are linear with dataset size. NNP is the fastest of all compared techniques, with KNNP using approximate nearest-neighbor (ANN) search coming close. Figure 13b adds *non-parametric* techniques to the comparison, specifically MDS [43], t-SNE, LSP [31], and LAMP [14]. We see the same trend as before. Also, we see that KNNP is faster than all non-parametric techniques. Separately, Figure 14 shows training time for the parametric techniques for up to 1M training samples. Beyond 250K samples, UMAP failed to finish training. NNP and KNNP with ANN search show basically the same speed, being both faster than KNNP with brute-force search.

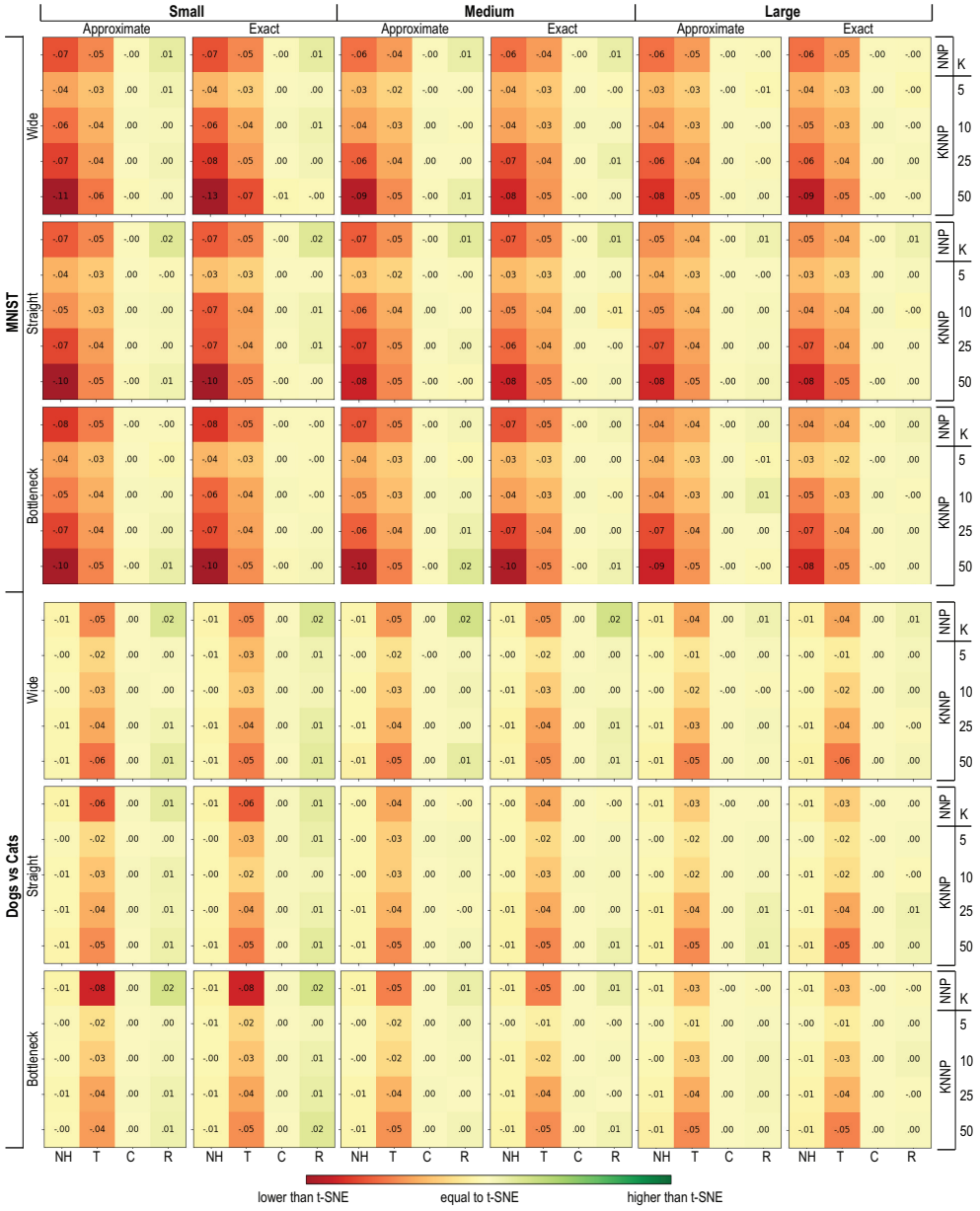


Fig. 10. Comparison of the difference in four quality metrics NH, T, C, and R between t-SNE and NNP, respectively t-SNE and KNNP. The comparison is done on the MNIST and Dogs *vs* Cats datasets, for five K values, using both exact and approximate search, for three architecture styles (wide, straight, bottleneck), each having three sizes (small, medium, large). Red colors indicate cases which are farthest below t-SNE's quality.

7.5 Projection scatterplots

Figure 15 shows samples of scatterplots created with t-SNE, NNP, and KNNP with ANN and brute force search. We see that KNN creates scatterplots which are less

Fig. 11. Comparison of quality metrics of KNNP *vs* NNP for the same datasets, architectures, and parameters as in Fig. 10. Green indicates cases where KNNP performs better than NNP.

fuzzy than NNP, being very close to the ones that t-SNE creates. For test data, note that both NNP and KNNP place point clusters at different locations than t-SNE. This is expected since, as explained, t-SNE is non-parametric. We also see that KNNP delivers visually identical plots for approximate *vs* exact search. Hence, we can use the faster approximate (ANN) search without fear of quality loss.

8 Discussion and conclusions

In this paper, we presented an in-depth study aimed at measuring and improving the quality of dimensionality reduction (DR) using deep learning supervised by existing DR techniques. We first explored the design space of a recent deep learning method for DR (NNP, [6]) along six dimensions: training-set size, network architecture, regularization, optimizers, data augmentation, and loss functions. We sampled each dimension using

Training Samples	KNN	NN Search	Train NH	Train T	Train C	Train R	Test NH	Test T	Test C	Test R
2k	5	Approx	.04	.01	.00	-.01	.03	.03	.00	-.01
		Exact	.04	.01	.00	-.02	.02	.03	.00	-.02
	10	Approx	.02	.00	.00	-.02	.00	.03	.00	-.01
		Exact	.04	.01	.00	-.01	.01	.02	.00	-.01
	25	Approx	-.01	-.02	.00	.00	-.02	.01	.00	.00
		Exact	-.06	-.03	.00	.01	-.03	.01	.00	.02
	50	Approx	-.06	-.04	.00	-.01	-.05	.00	.00	.00
		Exact	-.05	-.04	.00	.00	-.04	.00	.00	.02
5k	5	Approx	.00	.00	.00	-.01	.00	.01	.00	-.01
		Exact	.02	.01	.00	-.01	.01	.02	.00	-.01
	10	Approx	-.02	-.01	.00	-.01	-.01	.01	.00	.00
		Exact	-.01	-.01	.00	-.01	.00	.01	.00	.00
	25	Approx	-.03	-.02	.00	-.01	-.04	.00	.00	.00
		Exact	-.03	-.02	.00	.00	-.04	.00	.00	.01
	50	Approx	-.05	-.02	.00	.00	-.07	-.01	.00	.02
		Exact	-.04	-.02	.00	-.01	-.05	-.01	.00	.00
10k	5	Approx	.03	.02	.00	.00	.02	.02	.00	-.01
		Exact	.01	.01	.00	.00	.02	.02	.00	.00
	10	Approx	.02	.02	.00	.00	.02	.02	.00	.00
		Exact	.01	.01	.00	.00	.02	.02	.00	.00
	25	Approx	.00	.01	.00	.00	.00	.01	.00	.00
		Exact	-.01	.00	.00	.00	.00	.01	.00	.00
	50	Approx	-.02	.00	.00	.01	-.01	.01	.00	.02
		Exact	-.03	.00	.00	.00	-.02	.01	.00	.00

Training Samples	KNN	NN Search	Train NH	Train T	Train C	Train R	Test NH	Test T	Test C	Test R
2k	5	Approx	.00	.09	.00	-.03	.01	.06	.00	-.02
		Exact	.00	.09	.00	-.03	.00	.02	.00	-.01
	10	Approx	.00	.08	.00	-.03	.01	.06	.00	-.02
		Exact	.00	.08	.00	-.03	.00	.06	.00	-.02
	25	Approx	.00	.07	.00	-.03	.00	.04	.00	.00
		Exact	.00	.07	.00	-.03	.00	.04	.00	-.01
	50	Approx	.00	.04	.00	-.03	.00	.02	.00	-.01
		Exact	.00	.05	.00	-.03	.00	.03	.00	.00
5k	5	Approx	.00	.02	.00	-.01	.00	.03	.00	-.02
		Exact	.00	.02	.00	-.01	.00	.03	.00	-.02
	10	Approx	.00	.01	.00	-.01	.00	.02	.00	-.01
		Exact	.00	.02	.00	-.01	.00	.03	.00	-.01
	25	Approx	.00	-.01	.00	-.01	.00	.01	.00	-.01
		Exact	.00	-.01	.00	.00	.00	.01	.00	.00
	50	Approx	.00	-.03	.00	.00	.00	-.01	.00	.00
		Exact	.00	-.03	.00	.00	.00	-.01	.00	.00
10k	5	Approx	.00	.02	.00	-.01	.00	.03	.00	-.01
		Exact	.00	.02	.00	.00	.00	.03	.00	.00
	10	Approx	.00	.01	.00	-.01	.00	.03	.00	-.01
		Exact	.00	.01	.00	.00	.00	.03	.00	-.01
	25	Approx	.00	-.01	.00	-.01	.00	.02	.00	-.01
		Exact	.00	.00	.00	-.01	.00	.02	.00	-.01
	50	Approx	.00	-.02	.00	.00	.00	.01	.00	.00
		Exact	.00	-.02	.00	.00	.00	.01	.00	.00

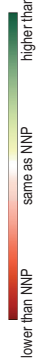


Fig. 12. Comparison of KNNP *vs* NNP quality metrics for different training set sizes on MNIST (left) and Dogs *vs* Cats (right). Green marks cases where KNNP outperforms NNP.

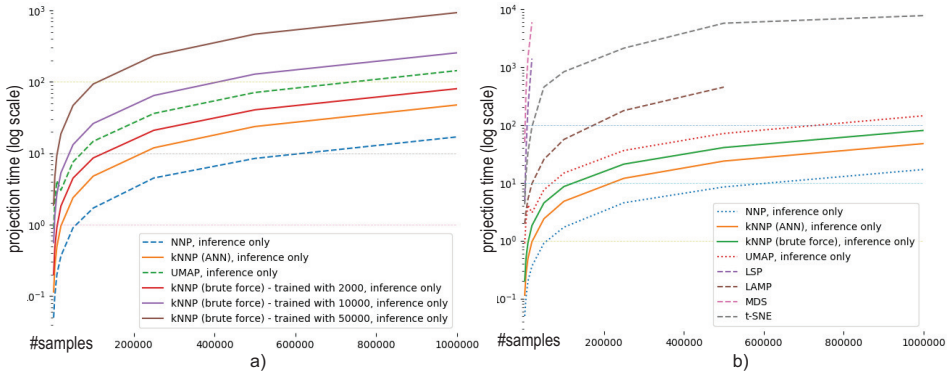


Fig. 13. Projection times for parametric techniques only (a) and for parametric and non-parametric techniques (b)

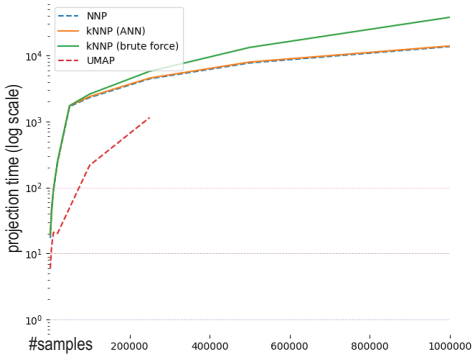


Fig. 14. Comparison of training times between parametric techniques

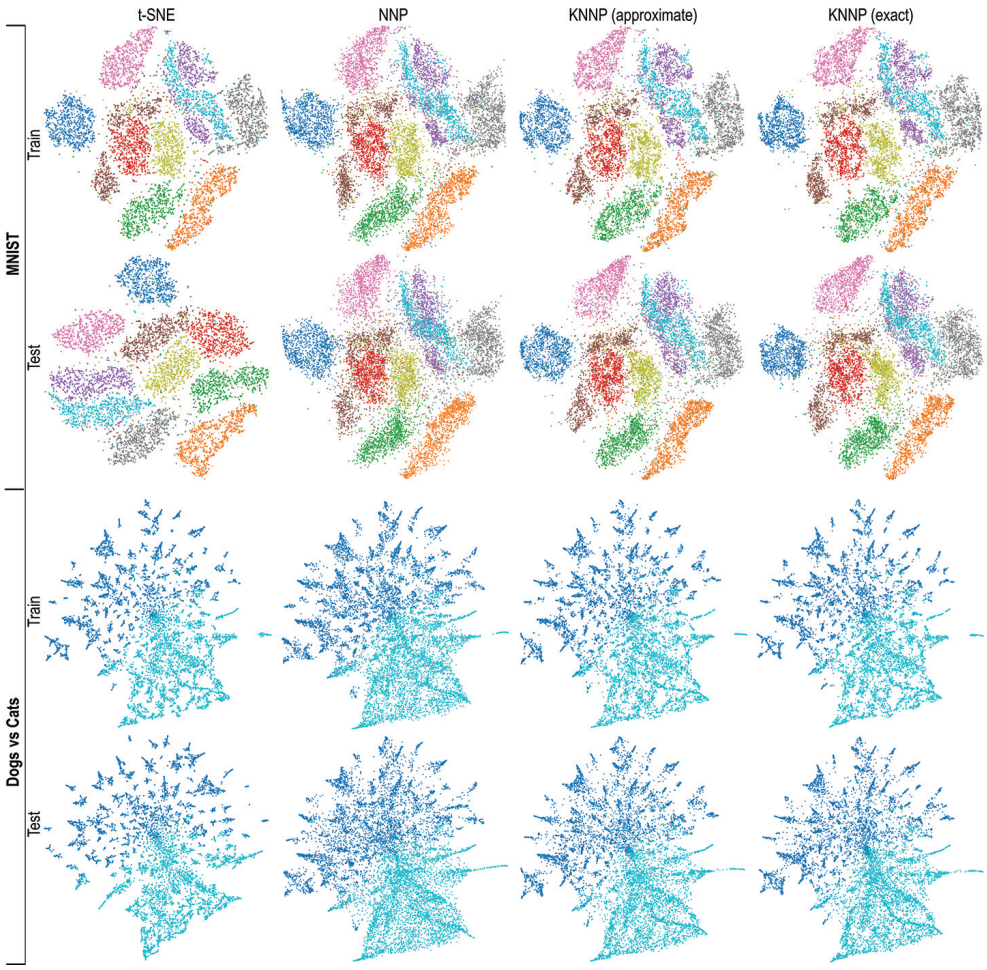


Fig. 15. Projections created by t-SNE, KNN, and KNNP (approximate and exact search variants) on MNIST and Dogs *vs* Cats datasets during training and testing (inference).

several method types and, where applicable, method parameter values, and compared the resulting NNP projections with the ground-truth (created by t-SNE) quantitatively, using four quality metrics, and also qualitatively by visual inspection of the respective scatterplots. Our exploration delivered an optimal hyperparameter setting that brings NNP closer to the quality of the t-SNE ground truth. Separately, we showed that NNP is stable with respect to all parameter settings, training-set size, and noise added to the input data.

Secondly, we further improved NNP quality by proposing KNNP, a refinement of the method that learns by projecting entire neighborhoods rather than individual samples. While improving quality, KNNP keeps the same attractive features of NNP, namely computational scalability, out-of-sample capability, and robustness to parameter settings. We also inferred optimal parameter settings for KNNP ($K = 5$ nearest neighbors found by approximate fast search) apart from the already established parameters it inherits from NNP discussed above.

Our results complement recent evaluations [6, 7] and show that supervised deep learning is a practical, robust, simple-to-set-up, and high-quality alternative to t-SNE for dimensionality reduction in data visualization. More broadly, we believe that our methodology can be directly used to reach the same goals (optimal settings and proof of stability) for any projection technique under study, whether this technique is using deep learning or not. We plan to extend these results in several directions. First, we aim to generalize the (K)NNP approach to work without the need of supervision given by a ground-truth projection. Secondly, we aim to extend (K)NNP to handle dynamic (time-dependent) high-dimensional datasets while keeping its aforementioned attractive points concerning stability, computational scalability, and ease of use.

References

1. Becker, M., Lippel, J., Stuhlsatz, A.: Regularized nonlinear discriminant analysis - an approach to robust dimensionality reduction for data visualization. In: Proc. VISIGRAPP. pp. 116–127 (2017)
2. Cunningham, J., Ghahramani, Z.: Linear dimensionality reduction: Survey, insights, and generalizations. JMLR **16**, 2859–2900 (2015)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Proc. CVPR. pp. 248–255 (2009)
4. Elson, J., Douceur, J.J., Howell, J., Saul, J.: Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In: Proc. ACM CCS. pp. 366–374 (2007)
5. Engel, D., Hüttenberger, L., Hamann, B.: A survey of dimension reduction methods for high-dimensional data analysis and visualization. In: Proc. IRTG Workshop. vol. 27, pp. 135–149. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2012)
6. Espadoto, M., Hirata, N., Telea, A.: Deep learning multidimensional projections. J. Information Visualization (2020), doi.org/10.1177/1473871620909485
7. Espadoto, M., Hirata, N.S.T., Falcão, A.X., Telea, A.C.: Improving neural network-based multidimensional projections. In: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP,. pp. 29–41. INSTICC, SciTePress (2020). https://doi.org/10.5220/0008877200290041
8. Espadoto, M., Martins, R.M., Kerren, A., Hirata, N.S., Telea, A.C.: Towards a quantitative survey of dimension reduction techniques. IEEE TVCG (2019), doi:10.1109/TVCG.2019.2944182
9. Feurer, M., Hutter, F.: Hyperparameter optimization. In: Automated Machine Learning. Springer (2019)
10. Goldberger, J., Roweis, S., Hinton, G.E., Salakhutdinov, R.R.: Neighbourhood components analysis. NIPS **17**, 513–520 (2005)

11. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
12. Hoffman, P., Grinstein, G.: A survey of visualizations for high-dimensional data mining. In: Information Visualization in Data Mining and Knowledge Discovery. pp. 47–82. Morgan Kaufmann (2002)
13. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.: Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates. In: Proc. AAAI (2017)
14. Joia, P., Coimbra, D., Cuminato, J.A., Paulovich, F.V., Nonato, L.G.: Local affine multidimensional projection. IEEE TVCG **17**(12), 2563–2571 (2011)
15. Jolliffe, I.T.: Principal component analysis and factor analysis. In: Principal Component Analysis, pp. 115–128. Springer (1986)
16. Kehrer, J., Hauser, H.: Visualization and visual analysis of multifaceted scientific data: A survey. IEEE TVCG **19**(3), 495–513 (2013)
17. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
18. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. CoRR **abs/1312.6114** (2013)
19. Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: NIPS. pp. 950–957 (1992)
20. LeCun, Y., Cortes, C., Burges, C.: MNIST handwritten digit database. AT&T Labs http://yann.lecun.com/exdb/mnist **2** (2010)
21. Liu, S., Maljovec, D., Wang, B., Bremer, P.T., Pascucci, V.: Visualizing high-dimensional data: Advances in the past decade. IEEE TVCG **23**(3), 1249–1268 (2015)
22. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. pp. 142–150. Association for Computational Linguistics (2011)
23. van der Maaten, L.: Learning a parametric embedding by preserving local structure. In: Proc. AI-STATS (2009)
24. van der Maaten, L.: Accelerating t-SNE using tree-based algorithms. JMLR **15**, 3221–3245 (2014)
25. van der Maaten, L., Hinton, G.E.: Visualizing data using t-SNE. JMLR **9**, 2579–2605 (2008)
26. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs (2016)
27. Martins, R., Minghim, R., Telea, A.C.: Explaining neighborhood preservation for multi-dimensional projections. In: Proc. CGVC. pp. 121–128. Eurographics (2015)
28. McInnes, L., Healy, J.: UMAP: Uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426 (2018)
29. Nonato, L., Aupetit, M.: Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. IEEE TVCG (2018)
30. Park, M.Y., Hastie, T.: L1-regularization path algorithm for generalized linear models. J Royal Stat Soc: Series B **69**(4), 659–677 (2007)
31. Paulovich, F.V., Nonato, L.G., Minghim, R., Levkowitz, H.: Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. IEEE TVCG **14**(3), 564–575 (2008)
32. Peason, K.: On lines and planes of closest fit to systems of point in space. Philosophical Magazine **2**(11), 559–572 (1901)
33. Pezzotti, N., Höllt, T., Lelieveldt, B., Eisemann, E., Vilanova, A.: Hierarchical stochastic neighbor embedding. Computer Graphics Forum **35**(3), 21–30 (2016)
34. Pezzotti, N., Lelieveldt, B., van der Maaten, L., Höllt, T., Eisemann, E., Vilanova, A.: Approximated and user steerable t-SNE for progressive visual analytics. IEEE TVCG **23**, 1739–1752 (2017)
35. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural networks **12**(1), 145–151 (1999)
36. Rauber, P., Falcão, A.X., Telea, A.: Visualizing time-dependent data using dynamic t-SNE. In: Proc. EuroVis: Short Papers. pp. 73–77 (2016)
37. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)

38. Salton, G., McGill, M.J.: Introduction to modern information retrieval. McGraw-Hill (1986)
39. Sorzano, C., Vargas, J., Pascual-Montano, A.: A survey of dimensionality reduction techniques. arXiv preprint arXiv:1403.2877 (2014)
40. Srebro, N., Shraibman, A.: Rank, trace-norm and max-norm. In: Intl. Conf. on Computational Learning Theory. pp. 545–560. Springer (2005)
41. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proc. CVPR. pp. 2818–2826 (2016)
42. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000)
43. Torgerson, W.: Theory and Methods of Scaling. Wiley (1958)
44. van der Maaten, L., Postma, E.: Dimensionality reduction: A comparative review. Tech. rep., Tilburg Univ., Netherlands (2009), tiCC 2009-005
45. Venna, J., Kaski, S.: Visualizing gene interaction graphs with local multidimensional scaling. In: Proc. ESANN. pp. 557–562 (2006)
46. Vernier, E., Garcia, R., da Silva, I., Comba, J., Telea, A.: Quantitative evaluation of time-dependent multidimensional projection techniques. Computer Graphics Forum **39**(20) (2020)
47. Wattenberg, M.: How to use t-SNE effectively (2016), `https://distill.pub/2016/misread-tsne`
48. Wilson, A., Roelofs, R., Stern, M., Srebro, N., Recht, B.: The marginal value of adaptive gradient methods in machine learning. In: NIPS, pp. 4148–4158. Curran Associates, Inc. (2017)
49. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
50. Xie, H., Li, J., Xue, H.: A survey of dimensionality reduction techniques based on random projection. arXiv preprint arXiv:1706.04371 (2017)
51. Yao, Y., Rosasco, L., Caponnetto, A.: On early stopping in gradient descent learning. Constructive Approximation **26**(2), 289–315 (2007)