Eurographics Conference on Visualization (EuroVis) 2022
R. Borgo, G. E. Marai, and T. Schreck
(Guest Editors)

Volume 41 (2022), Number 3

HyperNP: Interactive Visual Exploration of Multidimensional Projection Hyperparameters

G. Appleby[1], M. Espadoto[2,3], R. Chen[1], S. Goree[4], A. C. Telea[3], E. W. Anderson[5], and R. Chang[1]

[1]Tufts University, Department of Computer Science, USA
[2]University of Sao Paulo, Institute of Mathematics and Statistics, Brazil
[3]University of Utrecht, Department of Information and Computing Sciences, Netherlands
[4]Indiana University Bloomington, Luddy School of Informatics, Computing, and Engineering, USA
[5]Novartis AI Innovation Center. Novartis, USA

Abstract

Projection algorithms such as t-SNE or UMAP are useful for the visualization of high dimensional data, but depend on hyperparameters which must be tuned carefully. Unfortunately, iteratively recomputing projections to find the optimal hyperparameter values is computationally intensive and unintuitive due to the stochastic nature of such methods. In this paper we propose HyperNP, a scalable method that allows for real-time interactive hyperparameter exploration of projection methods by training neural network approximations. A HyperNP model can be trained on a fraction of the total data instances and hyperparameter configurations that one would like to investigate and can compute projections for new data and hyperparameters at interactive speeds. HyperNP models are compact in size and fast to compute, thus allowing them to be embedded in lightweight visualization systems. We evaluate the performance of HyperNP across three datasets in terms of performance and speed. The results suggest that HyperNP models are accurate, scalable, interactive, and appropriate for use in real-world settings.

CCS Concepts

• ***Human-centered computing*** → ***Visualization techniques;***

1. Introduction

As data-generating devices and computational resources expand, there is a growing need for interactive visual exploration for high-dimensional data [LMW*17]. An important class of visualization methods for such data is projection, which maps data from a high-dimensional space to a similarity-preserving low-dimensional representation [NA19, EMK*21]. Many projection methods exist, including linear and global techniques such as PCA [Pea01], and non-linear approaches such as t-SNE [vdMH08], UMAP [MHSG18], and Isomap [BS02].

While widely used to visualize high-dimensional data, projections can be sensitive to hyperparameter choices which the practitioner must tune carefully [SH05, WHRS21, WVJ16]. Unfortunately, many projection methods are computationally expensive [NA19], making real-time hyperparameter changes intractable. In order to support a user's exploration of hyperparameters in projections, a method is required that enables real-time interaction with a projection's hyperparameter values.

In this paper, we present HyperNP, a deep learning technique that approximates projections across hyperparameters to enable real-time exploration of a projection method's hyperparameters. Similar to previous work on neural network projection (NNP) [EHT20, EAS*21], HyperNP approximates the projection of a single technique (like t-SNE) on a single dataset with the added benefit of out-of-sample projection. Both NNP and HyperNP can be thought of as surrogate models for projection techniques, which attempt to alleviate some of the computational burden by approximating the results. Unlike previous methods, HyperNP can compute projections for hyperparameter values *unseen* during training, without the heavy recomputation inherent to the process for general projection methods. Previous methods [EHT20, EHFT20] have to refit each hyperparameter value since they only produce a single projection at a single hyperparameter value. HyperNP, on the other hand, produces projections across the range of the hyperparameter value of interest after a single training. Once HyperNP has been trained, the compute time required for the inference step is minimal. We leverage this fast computation to allow real-time visualization of large data sets and interactive exploration of different hyperparameter values associated with the projection operator.

As with most deep learning techniques, HyperNP has two stages: training and inference. Training HyperNP to mimic the outcome of a parameterized projection method has two steps: (1) The hyperparameters are sampled and used to project a subset of the data using the user-selected projection technique. (2) The HyperNP model is trained with a loss function that captures the differences between

its prediction and the ground-truth. After training, HyperNP infers the 2D projection of high-dimensional data for any user-selected projection hyperparameter value. At this stage, the user can interact with the hyperparameters using HyperNP, as the effect of these parameters can be estimated by the HyperNP model.

HyperNP is applicable to any projection requiring user-tuned input. This is noteworthy, as it means that the technique can be leveraged for interactive tuning of *any* projection method without reformulation. We demonstrate HyperNP's ability to learn hyperparameters by applying it to three popular techniques: t-SNE, UMAP, and Isomap. For each of these techniques, we explore one of their most important hyperparameters, and show how HyperNP can help users tune this hyperparameter to find an appropriate projection. This shows that HyperNP learns hyperparameter spaces that cover both continuous domains (t-SNE perplexity) as well as discrete integral domains (k-nearest neighbors value for UMAP and Isomap).

To evaluate HyperNP, we conducted experiments across three datasets to assess its (1) accuracy in approximating projection techniques across a variety of measures, (2) scalability, measured by training time, and (3) interactivity, measured by inference speed. The results demonstrate that HyperNP creates high-quality projections using a fraction of the training data and hyperparameter samples. Projections generated by HyperNP are visually similar to those produced by these projection methods, and perform similarly to the ground-truth in terms of common projection-quality metrics. Further, HyperNP is scalable with respect to the size of data, both in terms of training time and inference time. Training HyperNP with up to 50k data points requires less than 20 minutes. Once trained, HyperNP is fully interactive with 20ms latency for 50k data points and 500ms latency for up to 500k data points. In summary, we claim the following contributions: HyperNP, a deep learning-based **method** able to approximate projections of high-dimensional data across hyperparameter configurations at interactive speeds; **examples** of our method applied to three popular projection techniques (t-SNE, UMAP, and Isomap); and a quantitative **evaluation** of our method highlighting its accuracy, scalability, and interactivity.

2. Related Work

We start with some notation. Let $D = \{\mathbf{x}_i\}$, $\mathbf{x}_i \in \mathbb{R}^n$, $1 \leq i \leq N$ be a n-dimensional dataset. Its N points $\mathbf{x}_i$ (also called samples or observations) each have n dimensions (also called attributes).

2.1. Overview of Projection Techniques

Projection, or dimensionality reduction (DR) techniques, are members of a class of high-dimensional visualization methods. A DR method can be seen as a function $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$, where $q \ll n$ and, in the context of visualization of high-dimensional data, $q = 2$. The projection $P(\mathbf{x}_i)$ of a sample $\mathbf{x}_i \in D$ is thus a 2D point and the projection of D, denoted $P(D)$, can be shown as a 2D scatterplot. A projection P aims to place points $\mathbf{x}_i \in D$ that are regarded to be similar from the perspective of a measure defined on the data space D close to each other in the scatterplot $P(D)$. Hence, users can reason about data relationships in D by examining this scatterplot.

Projections scale well both in the number of samples N and dimensions n, enabling their wide use for visualizing high-dimensional data. Several works examine projections and their added-value from several viewpoints:

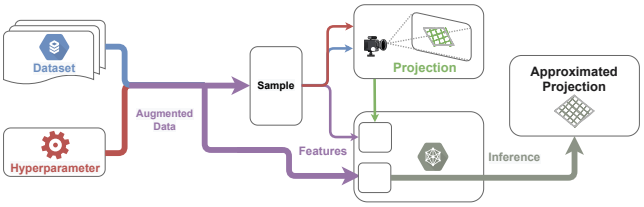


Figure 1: *Architecture of HyperNP: Training data from a dataset is projected across a range of a hyperparameters of interest belonging to a specific projection method. HyperNP's neural network is trained using a combination of a sample of the dataset to project and the projection methods's hyperparameters as features, and the projections of the dataset sample as target values. After training, HyperNP infers projections that approximate the original projection, but at a fraction of the computational cost. In the figure, line width relates to the number of observations; color shows when data are combined, split, or transformed.*

Taxonomies: Projection techniques can be organized into several taxonomies, thereby providing ways for practitioners to compare and choose a technique based on specific requirements. Van der Maaten *et al.* [vdMPvdH09] and Cunningham *et al.* [CG15] show how P can be computed by several types of non-linear, or respectively linear, optimization methods that offer different tradeoffs between computational scalability and projection quality. Sorzano *et al.* [SVM14] and Engel *et al.* [EHH12] propose two taxonomies of projections based on their implementation aspects, in particular the cost functions they optimize to compute P, and the choices of available optimizers and their computational complexities.

Quality and perception: A projection's effectiveness is a combination of how well $P(D)$ captures data patterns present in D and how (easily) users actually perceive the patterns in $P(D)$. Heulot *et al.* [HFA17] and Nonato and Aupetit [NA19] classify the different types of errors that projection techniques create and how these impact different classes of visual analytics tasks. Separately, several authors propose perception models [AEM11, TBB*10, WFC*18] and visual quality metrics and visualizations thereof [LV09, MLGH13, MCMT14, SA15, AS16, CHAS18] to help users understand the quality of the projections they compute, as determined by the chosen algorithms.

Quantitative analysis: Several works propose benchmarks which measure the quality of projection techniques across a selection of datasets [vdMPvdH09, EMK*21]. These provide guidelines for practitioners to choose a suitable technique depending on the type of quality metrics they want to maximize for a given dataset type.

2.2. Hyperparameters of Projection Techniques

The function P depends not only on D, but also on a set of *hyperparameters* $\mathbf{h} = \{h_i\}$. These hyperparameters control several aspects of P as follows. Computing a *global* mapping P between $\mathbb{R}^n$ and $\mathbb{R}^2$ that has overall high quality is, in general, not possible. As such, many projection techniques construct different mappings for different small-scale neighborhoods in $\mathbb{R}^n$. For example, in a number

of projection techniques (including UMAP, Isomap and others), a scalar parameter h controls the neighborhood size.

Locally Linear Embedding (LLE) [RS00] fits a hyperplane through each point and its nearest neighbors, keeping local relationships linear, but allowing the global structure to be nonlinear. Isomap [BS02] tackles the problem of projecting curved manifolds by estimating geodesic distances over neighborhoods and using these as a cost function to derive the projection. Least Squares Projection [PNML08a] projects a subset of landmarks and then uses a fast Laplacian-like operator to map the remaining samples. Two Phase Projection [PSN10], LAMP [JCC*11], and the Piecewise Laplacian Projection [PEP*11] use a similar approach. t-SNE [vdMH08] preserves neighbors during the projection by minimizing the Kullback-Leibler divergence between neighborhood probabilities in $\mathbb{R}^n$ and $\mathbb{R}^2$. UMAP [MHSG18] models the $\mathbb{R}^n$ manifold over small neighborhoods with a fuzzy topological structure and searches for a 2D representation that has the closest possible fuzzy topological structure. All these neighborhood-based approaches depend on hyperparameters $\mathbf{h}$ that model neighborhood sizes or number (and selection) of landmark points.

All of the discussed works mention that finding good hyperparameter values is challenging. Espadoto *et al.* [EMK*21] recognize this problem and address it by proposing optimal and preset parameter values computed by grid search over the hyperparameter space. However, this exhaustive search is expensive. More importantly, certain projection methods do not have *globally* optimal presets. Wattenberg *et al.* [WVJ16] illustrate this for t-SNE's perplexity parameter whose variation can create projections which emphasize different aspects of the n-dimensional data. However, experimenting with many hyperparameter values is a costly process, especially for projections that take a long time to compute.

Deep learning methods are effective for computationally scalable DR [HS06, SLZ12]. Kwon *et al.* [KM20] demonstrated success using a novel autoencoder to generate graph layouts. Their method enables the exploration of the latent space of graph layouts. More recently, NNP [EHT20, EHFT20, EAS*21] used deep learning to mimic any projection technique P by training on $P(D')$ for a small subset $D' \subset D$. In addition to providing fast computation, such approaches also have the ability to project *out-of-sample* data. Furthermore, due to the neural-network based approach, inference computations are parameter free. However, this is not always desirable. As observed by Wattenberg *et al.* [WVJ16], users may want to control $\mathbf{h}$ to gain insight into their data or to illustrate local versus global patterns (e.g. by changing the neighborhood size). Systems such as VisCoDeR [CHAS18] and t-viSNE [CMK20] allow users to explore how changes in hyperparameters affect projections, but their underlying projection methods do not offer the aforementioned benefits of NNP. HyperNP addresses the challenges of hyperparameter exploration and out-of-sample projection in a naturally scalable framework.

3. Method

We next detail HyperNP, a technique that can approximate any projection of a high-dimensional dataset at interactive speeds. This allows users to easily experiment with the hyperparameters of the approximated projection method. Similar to NNP [EHT20, EHFT20], we use a neural network to learn the approximation of a projection on a dataset. As NNP showed, such networks are accurate in their approximations and can infer a projection in a fraction of the time required by the learned technique. The configuration of the dense network used in this work is different from NNP and can be found in Section 3.4.

Figure 1 illustrates the process of using HyperNP to approximate a projection. Note that the dataset and the chosen hyperparameters are necessary for the creation of the ground truth projection and for training HyperNP. In addition to computing the projection $\mathbf{y} \in \mathbb{R}^2$ of a sample $\mathbf{x} \in \mathbb{R}^n$, and different from NNP, our trained network takes an additional argument $\mathbf{h}$ that represents the hyperparameters of the projection algorithm. For example, for UMAP, $\mathbf{h}$ is a single value that represents the number of nearest neighbors; for t-SNE, $\mathbf{h}$ is a single value that models perplexity. Performing inference with HyperNP on a data point $\mathbf{x}$ can be written as

$$\mathbf{y} = \hat{P}([\mathbf{x};\mathbf{h}]) \tag{1}$$

where $\hat{P}$ represents HyperNP's neural network, $\mathbf{y}$ is its projection of $\mathbf{x}$, and $[\mathbf{x};\mathbf{h}]$ is the concatenated vector of the sample and hyperparameter values that serves as input to $\hat{P}$.

3.1. Model Training in Theory

To train $\hat{P}$ to approximate a projection method P, we seek to minimize the difference between the projections output by $\hat{P}$ and P. Denoting $\hat{P}$'s parameters by θ, the objective can be expressed as

$$\arg\min_{\theta} \frac{\sum_{\mathbf{x}\in D}\sum_{\mathbf{h}\in H} \|\hat{P}_\theta([\mathbf{x};\mathbf{h}]) - P(\mathbf{x},\mathbf{h})\|^2}{|D|\cdot|H|}, \tag{2}$$

where D is the input dataset and H is the set of all values of $\mathbf{h}$.

We show empirically (Section 6) that training $\hat{P}$ does not require minimizing Eqn. 2 for all data points and across all values of $\mathbf{h}$. Depending on the algorithm P that $\hat{P}$ aims to approximate, we can use as little as 20% of all data points in D; for a hyperparameter $\mathbf{h}$, we can sample anywhere from every second to sixteenth value during training. We thus rewrite Eqn. 2 as

$$\arg\min_{\theta} \frac{\sum_{\mathbf{x}\in D'}\sum_{\mathbf{h}\in H'} \|\hat{P}_\theta([\mathbf{x};\mathbf{h}]) - P(\mathbf{x},\mathbf{h})\|^2}{|D'|\cdot|H'|}, \tag{3}$$

where $D' \subset D$ is the training set and $H' \subset H$ is the set of sampled values of the hyperparameters $\mathbf{h}$, respectively.

Consider a hyperparameter h ranging in some interval $[h_{min}, h_{max}]$, which is sampled $(h_{max} - h_{min})/g$ times. In the remainder of the paper, we refer to g as the '*gap size*'.

3.2. Model Training in Practice

Training P follows the typical workflow for training a neural network except for the generation of the training data, which must include the chosen hyperparameter values in addition to the raw features. To illustrate this, consider using HyperNP to explore UMAP's nearest-neighbor hyperparameter. In this example, we aim to train $\hat{P}$ using 20% of the data and a gap size of 6 to sample UMAP's neighborhood size $\mathbf{h}$ from 2 to 20. We begin generating the training data by randomly sampling 20% of the dataset D. We

then project the data four times for $\mathbf{h} \in \{2, 8, 14, 20\}$. We standardize the features and scale the resulting projections between 0 and 1. Once we have all of our projections, we stack the training data four times (once for each projection) and augment it by concatenating the $\mathbf{h}$ value used in the projection. Notably, the nearest neighbor value is not transformed. Finally, we train the network following usual procedures, using the augmented stacked matrix as input and the UMAP projections as target values. After training, the model can project the entire dataset across the full hyperparameter range.

3.3. Stability Considerations

Particular attention must be given to the issue of projection *stability*. Let P be a function of the hyperparameters $\mathbf{h}$ without considering dataset D. Then, we argue that $P(\mathbf{h})$ should be a relatively smooth function – small changes of $\mathbf{h}$ should yield only small changes in $P(\mathbf{h})$. Indeed, if this were not the case, P would confuse users by producing sudden jumps in the scatterplot when varying $\mathbf{h}$. This issue of stability *vs* hyperparameters $\mathbf{h}$ is analogous to the stability of projections *vs* changes in the data D, which was studied in detail for dynamic projections [VGS*20].

As discussed in [VGS*20], not all projection techniques are stable. We next identify and address two types of instabilities. The methods we use to address these instabilities are simple fixes that make this technique available to anyone and work well for our use case. Other solutions to this sort of stability problem can be found in these excellent surveys [TCL*13, ZGZ*19, HMZA21].

Seeding instabilities: Many stochastic techniques start from a random 2D scatterplot which is iteratively updated to minimize cost [vdMH08, JCC*11]. This makes P not smooth, since even with *no* parameter changes, P depends on the random seed (see *e.g.* the discussion in [EHT20], Figure 11). We address this as follows: Let $\mathbf{h}_i$ be the samples of the hyperparameter $\mathbf{h}$ used during training, in ascending order. We construct the training projection $P_i = P(D, \mathbf{h}_i)$ as usual, *i.e.*, running P with random seeding. Next, we construct P_{i+1} similarly, but use P_i to initialize the low-dimensional embedding of P_{i+1}. A similar strategy was used in [RFaT16] to construct stable t-SNE projections for dynamic datasets.

Eigenanalysis instabilities: Many projection techniques, (e.g. PCA [Pea01]), project data along eigenvectors which are agnostic towards dimension direction, or *sign*. One consequence of using such techniques is that the resulting projections can easily 'flip' along one of the 2D x or y axes upon slight changes in the input data or hyperparameters. Such drastic changes convey no meaning and should be eliminated. We address this problem using a similar solution to pose-invariance used in multimedia retrieval [BBFd07] for comparing arbitrarily rotated shapes. We compute the mean square error (MSE) between a training projection P_i and the previous projection P_{i-1} for all four possible mirrorings and pick the configuration with the lowest MSE for P_i. This minimizes undesired changes between P_i and P_{i-1} during training, making $\hat{P}$ more stable.

3.4. Implementation and Tuning

We implemented the dense neural network used by HyperNP with Tensorflow [AAB*15] and Keras [C*15]. We used KerasTuner's [OBL*19] Bayesian Optimization to find the appropriate number of layers, neurons per layer, and regularization. The tuning consisted of 512 trials, with the first 40 using random search as initial points. The search space included the number of layers (two to six), number of neurons per layer (32 to 512 in steps of 32), dropout probability [SHK*14] (0, 0.25, or 0.5), and whether or not to use batch normalization [IS15]. The final configuration used was a network with layer sizes $|V_1| = 320, |V_2| = 256, |V_3| = 352, |V_4| = 2$. We used ReLU [NH10] activation for all layers except the final one (V_4) where no activation function was used (linear). Between all layers we used batch normalization followed by a dropout probability of 0.25. We chose mean absolute error as the loss function and the ADAM [KB17] optimizer, following NNP [EHT20]. For more on the architecture of the network, see the supplemental material.

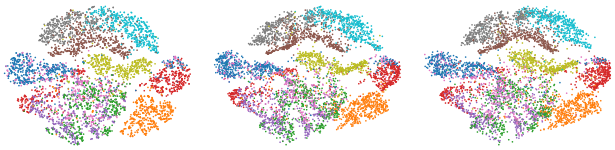
4. Data

We use three publicly available high-dimensional datasets that are reasonably large (thousands of samples) and have non-trivial data structure. **MNIST** [LCB10]: 70K images of handwritten digits from 0 to 9, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors; **FashionMNIST** [XRV17]: 70K images of 10 clothing piece types, rendered and flattened as for MNIST; **GloVe** [PSM14]: 70K GloVe (Global Vectors for Word Representation) vectors sampled from the 400k 300-dimensional vectors trained on Wikipedia 2014 + Gigaword 5.

5. Applications

We illustrate the value of HyperNP by showing how it approximates changes in the common hyperparameters provided by three popular projection methods: t-SNE, UMAP, and Isomap.

5.1. Exploring Perplexity in t-SNE



(a) *Ground Truth* **(b)** *Train Predictions* **(c)** *Test Predictions*

Figure 2: *HyperNP approximation of the t-SNE projections of the Fashion-MNIST dataset. From left to right: (a) t-SNE projection of 20% data that is then used to train HyperNP, (b) HyperNP projection of the same data used in (a), (c) HyperNP projection of test data instances unseen during training. This result suggests that HyperNP is adequately learning the t-SNE projection using just 20% of the data as these three images are visually similar.*

t-SNE is a variation of Stochastic Neighbor Embedding [HR03] which takes a probabilistic approach to preserving point-wise similarity when mapping high-dimensional data to a low-dimensional (typically 2D) space. First, a Gaussian distribution is constructed over the data samples so that similar sample pairs have a higher value than dissimilar ones. Another distribution is then constructed for the 2D space. t-SNE then places each sample in 2D by minimizing the Kullback-Leibler divergence between the two Gaussian Distributions. The bandwidth of the Gaussian kernels is chosen so

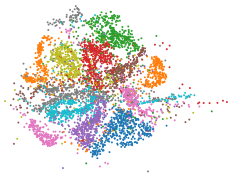
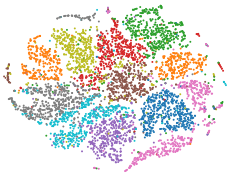
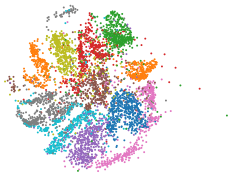
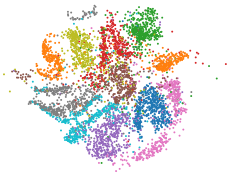
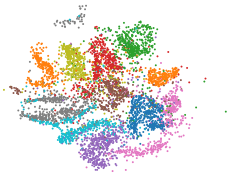
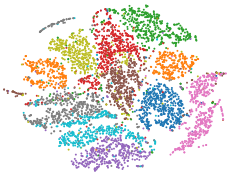
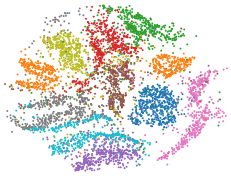
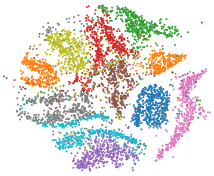
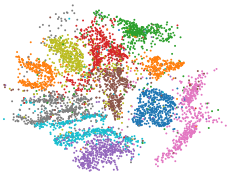
perplexity	t-SNE	HyperNP		
		Gap=2	Gap=4	Gap=8
5				
15				
25				

Table 1: t-SNE and HyperNP projections for three different perplexity values. First column: MNIST projected with t-SNE projections at perplexity values of $p = \{5, 15, 25\}$. The following three columns: HyperNP results when trained with gap values of 2, 4, and 8. First, as perplexity increases (within this range) in t-SNE, the global structures become more distinguishable. Further, notice that HyperNP can learn the t-SNE projections that result in visually similar plots to emulate the properties of perplexity, even when the hyperparameter is sampled sparsely (i.e. as the gap size increases from 2 to 8). All projections were trained using 40% of the original dataset.

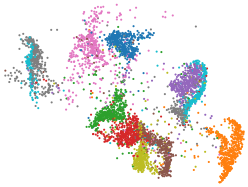
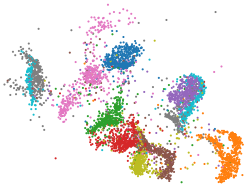
k	UMAP	HyperNP		
		Gap=2	Gap=4	Gap=8
5				
15				
25				

Table 2: UMAP and HyperNP projections for three different values of the nearest neighbors parameter. First column: MNIST projected with UMAP projections at nearest neighbor values of $k = \{5, 15, 25\}$. The following three columns: HyperNP results when trained with gap values of 2, 4, and 8. As the number of nearest neighbors increase in UMAP, the global structure is prioritized, losing some fine detail structure. HyperNP is able to capture these changes, properly capturing this-trade off even with a large sampling gap size. All projections were trained using 40% of the original dataset.

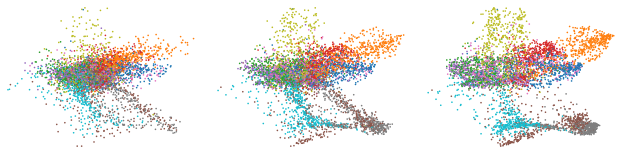
that the perplexity of the high dimensional distribution is equal to a user-defined *perplexity* hyperparameter. This adapts the size of the Gaussian kernel to the underlying density of the input data – smaller kernels are used in dense areas.

As Wattenberg *et al.* [WVJ16] point out, the value of the perplexity hyperparameter in t-SNE strongly affects the produced 2D embedding. Simply put, as perplexity increases, the importance of global features in the data also increases. Yet, the computational cost of robustly exploring the effects of many perplexity values is prohibitive for large data. Figure 2 shows how HyperNP can appropriately project out-of-sample data instances. This minimizes training time for each realized projection configuration, $\hat{P}([\mathbf{x};\mathbf{h}])$, allowing HyperNP to scale to large data with only moderate training cost.

While speed of both training and inference are important to the overall performance of HyperNP, care must be taken to ensure that the projection $\hat{P}$ is adequately learned. As Figure 2 shows, HyperNP has learned the t-SNE projection of the FashionMNIST dataset. The image created using only 20% of the data (randomly sampled) is similar to the others. This provides strong evidence that using only a small data fraction for training is enough to faithfully represent a chosen projection configuration.

Besides out-of-sample inference for *data*, HyperNP can perform out-of-sample inference for *hyperparameter* values. Table 1 shows how HyperNP explores different perplexity values for the MNIST dataset. Looking across rows, Table 1 shows that not only does HyperNP maintain similar clustering (columns 2-4) to the ground truth projection (column 1) for hyperparameters not seen in training, but also that increasing the gap size has only a marginal impact on the overall projection quality. This sampling strategy accelerates the initial training of the model without sacrificing quality.

Both Wattenberg *et al.* [WVJ16] and Silva *et al.* [ST02] argue for a thorough inspection of hyperparameters when interpreting projected data. Yet this critical task is often skipped due to its high computational cost. Not only does HyperNP provide the means to explore hyperparameters, but it does this for large datasets and without substantial quality loss. The increased interactivity allows users to better understand how the perplexity hyperparameter influences the final rendering of their data.



(a) *k=3* **(b)** *k=3.5* **(c)** *k=4*

Figure 3: *Using HyperNP to explore values of UMAP's nearest-neighbor hyperparameter, k, on FashionMNIST. From left to right: (a)* $k = 3.0$*, (b)* $k = 3.5$*, (c)* $k = 4.0$*. While non-natural values for the parameter k are not valid, we show that HyperNP is able to smoothly interpolate between meaningful values without sacrificing projection quality.*

5.2. Exploring k Nearest Neighbors in UMAP

UMAP is a projection technique based on ideas from topological data analysis and manifold learning. It is implemented using weighted graph techniques [MHSG18] derived from the notion of fuzzy simplicial sets. First, UMAP constructs a k-nearest-neighbor (kNN) graph, whose edge weights model the probability that the edge exists. The graph describes the locally connected manifold that the data is assumed to exist on, and to guide a force-directed graph layout in low dimensions. After initialization using spectral methods, a low dimensional graph is constructed from the projected points. The points are repositioned such that the low-dimensional graph approximates the original weighted graph in high dimension. The objective function minimized by the force directed layout is the total cross entropy of all edge probabilities, ensuring that the two graphs have similar topologies.

As with other nearest-neighbor based projections, a key hyperparameter of UMAP is k, the number of considered neighbors [WHRS21]. Varying k changes the local scale that the high dimensional manifold considers planar. Features that exist at smaller scale than the neighborhood size are thus lost, with larger features being better preserved in the projection. Silva *et al.* [ST02] provide a discussion of the importance of k with respect to feature size for projection techniques using manifold learning approaches.

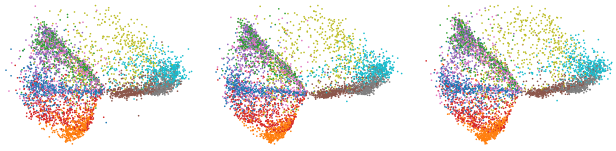
Typically, methods that use kNN graphs require all data to be integrated into the graph prior to projection. One challenge of manifold learning based projection techniques is the projection of out-of-sample data which does not exist in the kNN graph used to determine relationships across data points. HyperNP is able to project out-of-sample data without first embedding new points in the existing kNN graph. Table 2 shows how HyperNP maintains projection quality of out-of-sample data and also the importance of hyperparameter selection for different gap sizes: smaller neighborhoods lose projection quality faster as the gap size increases.

Care must be taken to ensure that the gap size used to train HyperNP does not overwhelm the smaller values for neighborhood size. The tradeoff between gap size and parameter lower bound is not unique to UMAP, but must be kept in mind for any manifold learning technique. As k changes in the low end of its range, the resulting change in the neighborhood topology is large: changing k from 3 to 4 yields a larger change than changing k from 20 to 21. Projection error increases as small values of k move away from the training k values, which is expected, as the projection change most rapidly for low k.

Unlike t-SNE's perplexity hyperparameter, $k \in \mathbb{N}$. Still, HyperNP is able to both learn and infer all $k \in \mathbb{R}$ values, as shown in Figure 3. That is, HyperNP can smoothly interpolate between the strict $k \in \mathbb{N}$ values.

5.3. Exploring Neighborhood Size in Isomap

Isomap is a technique that projects data points by first learning the manifold structure in high dimensions via kNN graphs [BS02]. As with UMAP, the most often changed hyperparameter of Isomap is the number of nearest neighbors to consider while building the kNN graph, k. However, unlike UMAP's notion of probabilistic edge



(a) *Ground Truth* **(b)** *Train Predictions* **(c)** *Test Predictions*

Figure 4: *HyperNP approximation of the Isomap projections of the FashionMNIST dataset. From left to right: (a) Isomap projection of 20% data that is then used to train HyperNP, (b) HyperNP projection of the same data used in (a), (c) HyperNP projection of data points unseen during training. This result suggests that HyperNP is adequately learning the Isomap projection using just 20% of the data as these three images are visually similar.*

weights (See Section 5.2), Isomap uses distances as edge weights. These weights enable methods like Dijkstra's Algorithm [Dij59] to approximate a geodesic distance along the induced manifold for each pair of points. After pair-wise geodesic distances are approximated, classical MDS methods are employed to construct the final low-dimensional embedding.

As both UMAP and Isomap make use of manifold learning and represent the manifold structure through kNN graphs, both methods rely on the value of the hyperparameter k to set the importance of global and local feature importance. Silva et al. provide a full discussion of the importance of this hyperparameter with respect to feature size for the class of projection techniques using manifold learning approaches [ST02]. As the underlying functionality of our method does not depend on any computational or theoretical framework unique to a given projection method, new methods are easily adapted to HyperNP's model. As shown in Figure 4, HyperNP can approximate Isomap with accuracy. These results are comparable to the same experiment performed using UMAP (See Figure 2). Additional attention must be paid to the value of k with respect to gap size in order to ensure projection quality remains high across all projection configurations for both UMAP and Isomap.

6. Evaluation

We evaluate HyperNP's performance through quantitative projection quality metrics and training and inferencing speed.

6.1. Performance: Quantitative Metrics

We measure the quality of HyperNP by the following metrics commonly used in the projection literature [EMK*21]. For more information on metrics and experiments, see the supplemental material.

Trustworthiness [VK06] measures the fraction of points in D that are also close in $P(D)$. It tells us how much we can trust the local patterns in a projection (*e.g.,* clusters) to represent actual data patterns. In the definition (Table 3), $U_i^{(K)}$ 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 point j in the ordered-set of nearest neighbors of i in 2D. We choose $K = 7$ in our experiments, in line with [vdMPvdH09, MMT15, EMK*21].

Continuity [VK06] measures the fraction of close points in $P(D)$ that are also close in D. In the definition (See Table 3), $V_i^{(K)}$ 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 point j in the ordered set of nearest neighbors of i in $\mathbb{R}^n$. Similar to trustworthiness we choose $K = 7$ following convention.

Hit Rate [PNML08b] measures the proportion of the neighbors of a point in a projection that have the same label as the point. In the definition (See Table 3), $W_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in 2D that share the same label as point i. For ease of computation we calculate hit rate on a uniform sampling of 10% of the data, unlike the previous two metrics.

Metric	Definition	Range
Trustworthiness	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in U_i^{(K)}} (r(i,j) - K)$	$[0, \mathbf{1}]$
Continuity	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in V_i^{(K)}} (\hat{r}(i,j) - K)$	$[0, \mathbf{1}]$
Hit Rate	$\frac{1}{N} \sum_{i=1}^{N} \lvert W_i^K \rvert / K$	$[0, \mathbf{1}]$

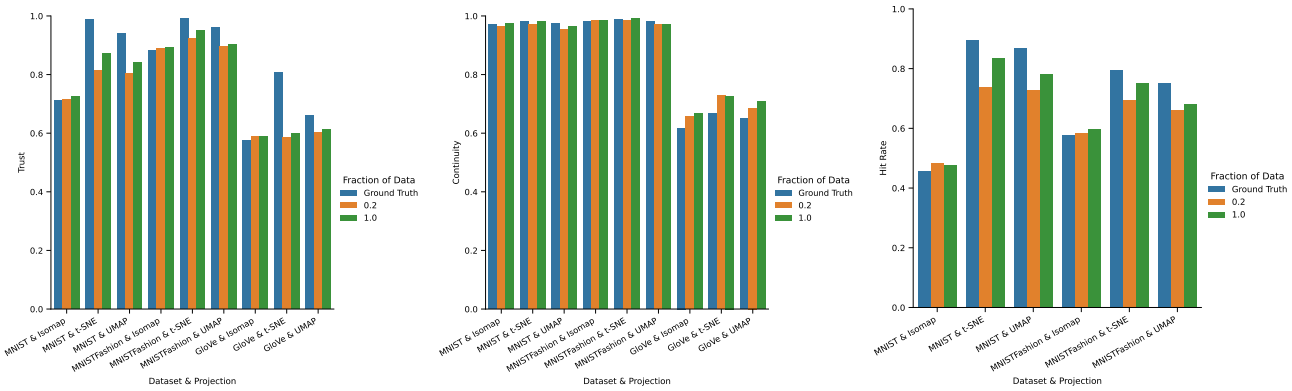
Table 3: Projection quality metrics used in our evaluation with optimal values in bold in the Range column.

We next explore different hyperparameter settings to determine their effect on HyperNP's quality, namely the gap size and the data fraction used for training (see Section 3). We first examine the metrics in aggregate, by taking their averages across hyperparamter values. For each dataset and projection combination, we fix the gap size and training fraction and compute the mean trustworthiness and continuity values across either perplexity or k, from two to fifty. We present these results in Figure 5. Notice that across most dataset and projection combinations within the figure, our method produces trustworthiness, continuity, and hit rate scores within a few percentage points of the ground truth projection method. Decreasing the size of the training data by decreasing data fraction from 1.0 to 0.2 (orange bars versus blue bars in Figure 5), a 5-fold decrease, shows only a minimal loss in scores with the benefit of decreased training time. These results inspire confidence; they show that HyperNP is able to infer projection locations of out-of-sample data, generalizing from a small subset of the entire dataset.

In Figure 6, we plot trustworthiness and continuity for UMAP and HyperNP trained with UMAP, varying the value of the hyperparameter k. We further vary the data fraction and gap size used to train HyperNP. In this experiment, HyperNP performs similarly to the ground truth UMAP for continuity, and at worst within approximately 85% of hit rate and 80% of trustworthiness. The relatively similar continuity scores, and different trust scores suggest that approximated projections have the same level of false neighbors as the real projections, but have more missing neighbors. False neighbors are points close in low dimensional space that were not close in the high dimensional space, while missing neighbors are the opposite [NA19].

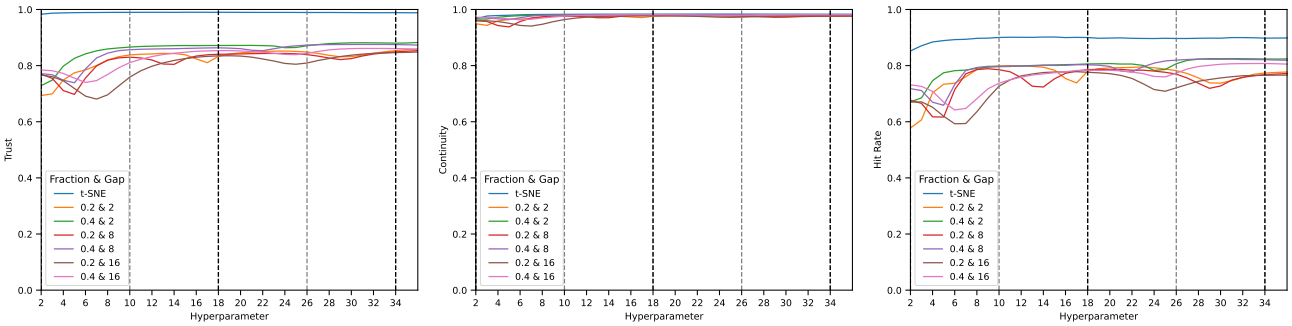
6.2. Performance: Training and Inferencing Speeds

Training Speed: Figure 7 shows training times for HyperNP for different gap sizes and data fraction values. The figure illustrates the trade-offs in training speed for different data fractions and gap sizes. As detailed in Section 3, the neural network will be trained on



(a) *Trustworthiness* **(b)** *Continuity* **(c)** *Hit Rate*

Figure 5: *In this figure, we present the trustworthiness, continuity, and hit rate scores associated with the ground truth projections as well as HyperNP trained using a gap size of* 16*, and data fractions of* 0.2 *and* 1.0*. We have averaged the scores over all of the hyperparameter values from* 2 *to* 50*. Comparing the ground truth score to HyperNP's score for each dataset and projection combination illustrates that HyperNP is able to perform close to the ground truth even under extreme sampling conditions (i.e., when only 20% of the data is used for training).*



(a) *Trustworthiness* **(b)** *Continuity* **(c)** *Hit Rate*

Figure 6: *Trustworthiness, continuity, and hit rate for HyperNP trained with t-SNE, on the MNIST dataset, for different hyperparameter values. Light gray vertical lines show parameter values used for training with a gap size of 8; dark gray ones show parameter values used for training with a gap size of 16. This dataset and projection combination yielded the greatest difference in trust scores between ground truth and HyperNP out of all of the combinations tested. Despite this, quality does not decrease substantially even during the dips far from sampled points. It is worth noting that some visual samples from the purple line representing a gap of 8 and a data fraction of .4 can be in the last column of Table 2.*

the number of sampled instances times the number of hyperparameters. For our experiments we chose what we expect an average case would be, examining hyperparameter values from 2 to 50. It should be noted that Figure 7 shows overall training time for HyperNP as a function of dataset size. The number of training instances is determined not just by this value, but the number of hyperparameter values and fraction of this number used.

As shown in Figure 7, in extreme cases when the gap size is too low, training HyperNP could take up to 80 minutes for a dataset with 50,000 points. However, as seen in Section 6.1 using a data fraction value of 0.4 (i.e., sampling rate of 40%) and gap size of 8 produces HyperNP projections of reasonable quality. With these configurations, most datasets with less than 50,000 points can be trained within 20 minutes. The time to train HyperNP in Figure 7 includes the time to project the ground truth data using Multicore-TSNE [Uly16] with four cores.

The HyperNP model is trained with a batch size of 32 in Figure 7. We note that the batch size will often have a considerable effect on not only the wall time it will take a network to converge, but also the final performance of the model. A more thorough discussion of this topic can be found in a number of papers [KMN*17, ML18, HLT19]. The training speed tests were executed on a 2-way, 16-core Intel Xeon Silver 4216@2.1 GHz with 256 GB of RAM, and an NVidia GeForce 1080 Ti GPU.

Inferencing Speed: We evaluate the inferencing speed of HyperNP. Table 4 shows the HyperNP's inference run-time using a batch size of 20k instances. We see that HyperNP remains interactive with 500k data points, performing inference under 500ms [LH14]. Overall, the speed of HyperNP scales linearly with the number of data points projected. The inferencing speed tests were executed on a 8-core Intel i7-6700k@4 GHz with 64 GB of RAM, and an NVidia GeForce 1080 GPU, with 8GB of VRAM.

Number of Instances	Seconds
56,000	0.192241
168,000	0.264158
336,000	0.370889
504,000	0.479788
616,000	0.551125
728,000	0.624992
840,000	0.695622
952,000	0.766805
1,064,000	0.839778

Table 4: HyperNP's inference speed

7. Using HyperNP to Approximate iPCA

Beyond approximating projection methods, HyperNP can have additional applications in visual analytics given its use of a neural network. In this section, we describe how HyperNP can be used to approximate iPCA [JZF*09], an interactive visual analytics technique for supporting exploration of high-dimensional data with PCA. iPCA is a technique that allows users to interactively adjust the weighted contributions of data dimensions and observe the effects in a PCA projection. Similar to HyperNP, iPCA allows a user to interactively manipulate projections using sliders in real time.

Figure 8 shows HyperNP's approximation of iPCA. The training of HyperNP is the same as described in Section 3. In Eqn. 1, **h** now is a n-dimensional vector that represents the scaling factors in iPCA where n is the number of data dimensions, each controlled by a slider in iPCA. We see that HyperNP is successful in emulating iPCA: interactions with the sliders in HyperNP produce the same projections as iPCA while retaining the interactive speed. This is important in terms of genericity of HyperNP. iPCA is designed specifically around PCA, using Online SVD [Bra03] to reduce the computation cost of singular value decomposition from $O(N^2)$ to $O(n^2)$ for N samples of n dimensions. As a result, it is hard to replace iPCA with another projection technique while maintaining interactivity. In contrast, doing so with HyperNP is simple – replacing PCA by its HyperNP approximation requires just changing the function P in Eqn. 3 from PCA to another projection technique. This supports the use of HyperNP to generalize high-dimensional data exploration techniques (like iPCA) beyond their associated projection methods. As future work, we will investigate other interaction techniques in visual analytics that can be made more generalizable with the use of HyperNP.

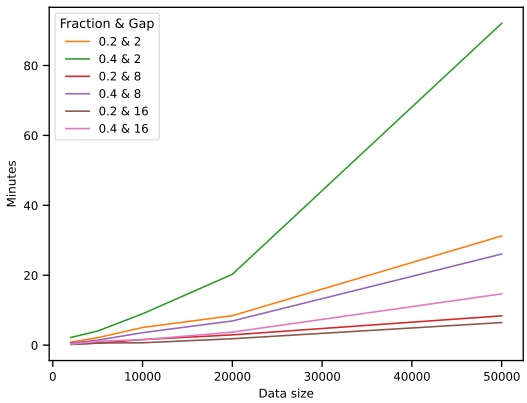


Figure 7: *HyperNP training speed with the MNIST dataset and t-SNE training projection using different amounts of training data (data fractions of 0.2 and 0.4) and gap sizes (2, 8, and 16). Smaller gap sizes result in larger* |**h**| *and thus longer training times. For smaller datasets (less than 20k), HyperNP can be fully trained within 10 to 20 minutes.*

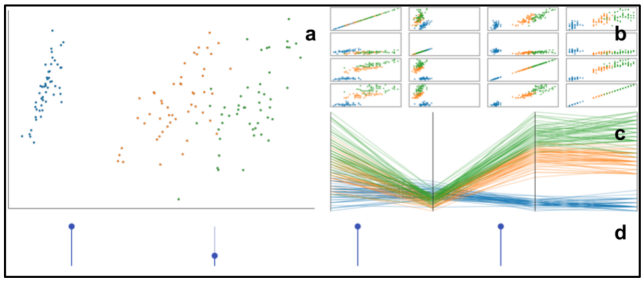


Figure 8: *A HyperNP implementation of iPCA. Each slider (d) corresponds to a dimension of the input dataset (Iris) [And35]. Adjusting a slider scales a feature from 100% to 0%. This is reflected in the parallel coordinates plot (c) and scatterplot matrix (b). HyperNP re-projects (a) the dataset in real time as the user interacts with the sliders (Section 6.2).*

8. Discussion, Limitations, and Future Work

In this section, we discuss both conceptual and practical considerations of using HyperNP. We conclude by presenting the limitations of HyperNP and some possible future research directions.

8.1. Sampling and Exploring Hyperparameters

This work shows HyperNP is able to approximate important hyperparameters associated with a projection technique. Exploring projection hyperparameters is exemplified in Tables 1 and 2. These images highlight how HyperNP captures much of the structure of the original projections using only a fraction of the data for training of the system. Similar to interpolation, increasing the gap size during HyperNP training introduces additional deviation from ground truth. These errors are most evident with low values of perplexity and k for tSNE and UMAP, respectively. At these small hyperparameter values, the number of samples naturally defining the local structure in the embedding space is likely insufficient to provide enough signal for HyperNP to approximate robustly. However, as the hyperparameter values increase, the local topology captured by the methods is less influenced by small changes to the hyperparmeter, allowing HyperNP to better approximate the projection, even when using larger gap sizes.

8.2. Conceptual and Practical Considerations of HyperNP

Using deep learning to approximate projections raises some important considerations. Since a neural network does not consider the semantics of a hyperparameter, it can generate projections using invalid configurations, see *e.g.* the use of non-integral k values for UMAP (Section 5.2). In this case, HyperNP infers the projection by interpolating the hyperparameter setting between integral values

used during training. While formally, this goes beyond the assumptions of UMAP (*i.e.*, k is an integer), this interpolation has yielded reasonable results for all k values, with no negative consequences noticed (see Figure 3).

The hyperparameter sampling strategy is also an important factor for HyperNP. While t-SNE's perplexity and UMAP's k-nearest neighbors are both hyperparameters, they respond differently to sampling. Figure 6 shows how quality changes when hyperparameters move farther from training values. For t-SNE, neither continuity nor trustworthiness change much. Still, while these metrics show a dip in the middle of large gaps (farthest from k training values), we still get high quality especially past the smaller hyperparameter values. This tells us that the gap size used to train HyperNP has greater impact for UMAP than for t-SNE; yet, even a gap size of 16 only marginally changes the projection quality. Summarizing, choosing the sample step (gap size) of hyperparameters needs to be studied individually for different projection techniques.

8.3. Limitations and Future Work

In this section we present potential limitations of HyperNP and discuss avenues to further develop this work.

Multiple Hyperparameters: Section 6 discusses the training time for HyperNP for a single hyperparameter. This time can grow exponentially as the number of hyperparameters $|\mathbf{h}|$ increases. If the complexity of training a traditional network is $O(\mathrm{K})$, the complexity of training HyperNP with one hyperparameter is $O(\mathrm{K} \cdot |H'|)$ with $|H'|$ as described in Eqn. 3. For $|\mathbf{h}|$ hyperparameters, this becomes $O(\mathrm{K} \cdot |H'|^{|\mathbf{h}|})$. Further work is needed to develop methods to reduce the training cost of HyperNP for projection algorithms with many hyperparameters. Additionally, an interface for intuitively exploring all hyperparameters associated with a projection method would need to be developed.

Hyperparameter Selection: Due to the current limitation of HyperNP in learning multiple hyperparameters, the user must carefully choose the most important hyperparameter to explore. In this paper we select the hyperparameters that affect the tradeoff between global and local preservation as the user's primary exploration dimension, following conventions established by Wattenberg et al. [WVJ16]. Prior work like VisCoDeR [CHAS18] and t-viSNE [CMK20] allow users to explore the effect of other hyperparameters that can affect the quality of the projections. We leave the evaluation of HyperNP on other hyperparameters for future work.

Training Data: Like any deep learning technique, HyperNP is influenced by the quality of training data. As detailed in Section 6.1, performance on the GloVe dataset is lower than the other two datasets, but performance relative to the ground truth projections is still high. HyperNP faithfully reconstructs the training data; however, the quality of a projection is directly related to the quality of the initial projections used during training. Even if the training data as a whole is of good quality, it is possible to introduce bias through sampling. Unintentional bias is introduced by any sampling strategy (to create training sets) that results in a different distribution than the dataset at large. We note that the problem of sampling training data is not unique to our method. We leave as future work strategies to mitigate its effects on approximating projections.

Training data quality is important when considering performance in the context of outliers and large data. Datasets containing outliers pose the same challenges for HyperNP as they do for the both the underlying projection technique being used and the deep learning architecture learning the projection. Exploring novel methods to mitigate the impact of outliers is beyond the scope of this work, but advances in this topic are easily integrated with the system. Since HyperNP is, at its core, a deep learning implementation, increasing the dataset size is generally considered to be an important step to improve performance. However, as discussed above, maintaining a similar distribution in the training data with respect to the overall dataset is important and must be considered when deciding when to update the HyperNP model representing the projection operator.

Smoothness Perception: The technique described in Section 3.3 maintains projection stability and visual coherence between frames as users interactively change a hyperparameter value. When combined with the high interactivity of HyperNP, this may induce a false sense of continuity concerning the *properties of the underlying learned projection method*. For example, consider approximating UMAP with HyperNP (Section 5.2). When the hyperparameter k changes from k to $k+1$, there is no guarantee that the two learned manifolds share correspondences, so the perception of smoothness created by HyperNP may be misleading. Conversely, for iPCA (Section 7), weight changes should result in a smooth animation as PCA rotates the underlying coordinate system to maximize the projected data variance. In summary, the built-in perception of smoothness of HyperNP should not always be interpreted as reflecting mathematical smoothness properties of the learned projections. Additionally, the neural network architecture learning the projection operator enables the use of potentially unintended parameter values. For example, a non-integer value of k in the UMAP operator is possible to approximate using HyperNP. Of course, the interpretation of a non-integer number of neighbors is challenging.

9. Conclusion

We proposed HyperNP, a deep learning approach to approximating projections that enables real-time interactive hyperparameter exploration. We showed that HyperNP can learn to infer projections of several techniques, for a wide range of parameter values, and different real-world datasets, using only a small fraction of the data and hyperparameter values. HyperNP performs in real-time for reasonably large datasets, can be generalized to any projection technique (and hyperparameters), and produces stable animations of the resulting projections upon parameter variations. We demonstrated HyperNP through the exploration of the key hyperparameters of t-SNE, UMAP, and Isomap.

10. Acknowledgements

This work was supported by National Science Foundation grants IIS1452977, OAC-1940175, OAC-1939945, DGE-1855886, OAC-2118201, and DOD grants HQ0860-20-C-7137, N68335-17-C-0656. We would also like to thank the reviewers for their helpful feedback.

References

[AAB*15] ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y., ZHENG X.: TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/. 4

[AEM11] ALBUQUERQUE G., EISEMANN M., MAGNOR M.: Perception-based visual quality measures. In *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology* (2011), pp. 13–20. doi:10.1109/VAST.2011.6102437. 2

[And35] ANDERSON E.: The irises of the gaspe peninsula. *Bulletin of the American Iris Society* (1935), 2–5. 9

[AS16] AUPETIT M., SEDLMAIR M.: Sepme: 2002 new visual separation measures. In *Proceedings of the 2016 IEEE Pacific Visualization Symposium* (2016), pp. 1–8. doi:10.1109/PACIFICVIS.2016.7465244. 2

[BBFd07] BLANKEN H., BLOK H., FENG L., DE VRIES A. (Eds.): *Multimedia Retrieval*. Data-Centric Systems and Applications. Springer, Netherlands, 2007. doi:10.1007/978-3-540-72895-5. 4

[Bra03] BRAND M.: Fast online svd revisions for lightweight recommender systems. In *Proceedings of the 2003 SIAM International Conference on Data Mining* (2003), pp. 37–46. doi:10.1137/1.9781611972733.4. 9

[BS02] BALASUBRAMANIAN M., SCHWARTZ E. L.: The isomap algorithm and topological stability. *Science 295*, 5552 (2002), 7–7. doi:10.1126/science.295.5552.7a. 1, 3, 6

[C*15] CHOLLET F., ET AL.: Keras. https://keras.io, 2015. 4

[CG15] CUNNINGHAM J. P., GHAHRAMANI Z.: Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research 16*, 89 (2015), 2859–2900. URL: http://jmlr.org/papers/v16/cunningham15a.html. 2

[CHAS18] CUTURA R., HOLZER S., AUPETIT M., SEDLMAIR M.: VisCoDeR: A tool for visually comparing dimensionality reduction algorithms. In *Proceedings of the 26th European Symposium on Artificial Neural Networks* (2018). 2, 3, 10

[CMK20] CHATZIMPARMPAS A., MARTINS R. M., KERREN A.: t-viSNE: Interactive assessment and interpretation of t-sne projections. *IEEE Transactions on Visualization and Computer Graphics 26*, 8 (2020), 2696–2714. doi:10.1109/TVCG.2020.2986996. 3, 10

[Dij59] DIJKSTRA E. W.: A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (Dec 1959), 269–271. doi:10.1007/BF01386390. 7

[EAS*21] ESPADOTO M., APPLEBY G., SUH A., CASHMAN D., LI M., SCHEIDEGGER C. E., ANDERSON E. W., CHANG R., TELEA A. C.: UnProjection: Leveraging inverse-projections for visual analytics of high-dimensional data. *IEEE Transactions on Visualization & Computer Graphics*, 01 (2021), 1–1. doi:10.1109/TVCG.2021.3125576. 1, 3

[EHFT20] 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* (2020), pp. 29–41. doi:10.5220/0008877200290041. 1, 3

[EHH12] ENGEL D., HÜTTENBERGER L., HAMANN B.: A Survey of Dimension Reduction Methods for High-dimensional Data Analysis and Visualization. In *Proceedings of the Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - IRTG 1131 Workshop 2011* (2012), vol. 27 of *OpenAccess Series in Informatics*, pp. 135–149. doi:10.4230/OASIcs.VLUDS.2011.135. 2

[EHT20] ESPADOTO M., HIRATA N. S. T., TELEA A. C.: Deep learning multidimensional projections. *Information Visualization 19*, 3 (2020), 247–269. doi:10.1177/1473871620909485. 1, 3, 4

[EMK*21] ESPADOTO M., MARTINS R. M., KERREN A., HIRATA N. S. T., TELEA A. C.: Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics 27*, 3 (2021), 2153–2173. doi:10.1109/TVCG.2019.2944182. 1, 2, 3, 7

[HFA17] HEULOT N., FEKETE J.-D., AUPETIT M.: Visualizing dimensionality reduction artifacts: An evaluation, 2017. arXiv:1705.05283. 2

[HLT19] HE F., LIU T., TAO D.: Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In *Proceedings of the Advances in Neural Information Processing Systems* (2019), vol. 32. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a70712a252123c40d2adba6a11d84-Paper.pdf. 8

[HMZA21] HUANG X., MEI G., ZHANG J., ABBAS R.: A comprehensive survey on point cloud registration, 2021. arXiv:2103.02690. 4

[HR03] HINTON G. E., ROWEIS S.: Stochastic neighbor embedding. In *Proceedings of the Advances in Neural Information Processing Systems* (2003), vol. 15. URL: https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf. 4

[HS06] HINTON G. E., SALAKHUTDINOV R. R.: Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507. doi:10.1126/science.1127647. 3

[IS15] IOFFE S., SZEGEDY C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning* (2015), JMLR.org, p. 448–456. 4

[JCC*11] JOIA P., COIMBRA D., CUMINATO J. A., PAULOVICH F. V., NONATO L. G.: Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2563–2571. doi:10.1109/TVCG.2011.220. 3, 4

[JZF*09] JEONG D. H., ZIEMKIEWICZ C., FISHER B., RIBARSKY W., CHANG R.: iPCA: An interactive system for pca-based visual analytics. *Computer Graphics Forum 28*, 3 (2009), 767–774. doi:https://doi.org/10.1111/j.1467-8659.2009.01475.x. 9

[KB17] KINGMA D. P., BA J.: Adam: A method for stochastic optimization, 2017. arXiv:1412.6980. 4

[KM20] KWON O.-H., MA K.-L.: A deep generative model for graph layout. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2020), 665–675. doi:10.1109/TVCG.2019.2934396. 3

[KMN*17] KESKAR N. S., MUDIGERE D., NOCEDAL J., SMELYANSKIY M., TANG P. T. P.: On large-batch training for deep learning: Generalization gap and sharp minima, 2017. arXiv:1609.04836. 8

[LCB10] LECUN Y., CORTES C., BURGES C.: MNIST handwritten digit database. *ATT Labs [Online] 2* (2010). URL: http://yann.lecun.com/exdb/mnist. 4

[LH14] LIU Z., HEER J.: The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2122–2131. doi:10.1109/TVCG.2014.2346452. 8

[LMW*17] LIU S., MALJOVEC D., WANG B., BREMER P., PASCUCCI V.: Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics 23*, 3 (2017), 1249–1268. doi:10.1109/TVCG.2016.2640960. 1

[LV09] LEE J. A., VERLEYSEN M.: Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing 72*, 7 (2009), 1431–1443. Advances in Machine Learning and Computational Intelligence. doi:10.1016/j.neucom.2008.12.017. 2

[MCMT14] MARTINS R. M., COIMBRA D. B., MINGHIM R., TELEA A.: Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics 41* (2014), 26–42. doi:10.1016/j.cag.2014.01.006. 2

[MHSG18] MCINNES L., HEALY J., SAUL N., GROSSBERGER L.: Umap: Uniform manifold approximation and projection. *Journal of Open Source Software 3*, 29 (2018), 861. doi:10.21105/joss.00861. 1, 3, 6

[ML18] MASTERS D., LUSCHI C.: Revisiting small batch training for deep neural networks, 2018. arXiv:1804.07612. 8

[MLGH13] MOKBEL B., LUEKS W., GISBRECHT A., HAMMER B.: Visualizing the quality of dimensionality reduction. *Neurocomputing 112* (2013), 109–123. Advances in artificial neural networks, machine learning, and computational intelligence. doi:10.1016/j.neucom.2012.11.046. 2

[MMT15] MARTINS R. M., MINGHIM R., TELEA A. C.: Explaining Neighborhood Preservation for Multidimensional Projections. In *Computer Graphics and Visual Computing* (2015), Borgo R., Turkay C., (Eds.), The Eurographics Association. doi:10.2312/cgvc.20151234. 7

[NA19] NONATO L. G., AUPETIT M.: Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE Transactions on Visualization and Computer Graphics 25*, 8 (2019), 2650–2673. doi:10.1109/TVCG.2018.2846735. 1, 2, 7

[NH10] NAIR V., HINTON G. E.: Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (2010), ICML'10, p. 807–814. 4

[OBL*19] O'MALLEY T., BURSZTEIN E., LONG J., CHOLLET F., JIN H., INVERNIZZI L., ET AL.: Kerastuner. https://github.com/keras-team/keras-tuner, 2019. 4

[Pea01] PEARSON K.: LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2*, 11 (1901), 559–572. doi:10.1080/14786440109462720. 1, 4

[PEP*11] PAULOVICH F., ELER D., POCO J., BOTHA C., MINGHIM R., NONATO L.: Piece wise laplacian-based projection for interactive data exploration and organization. *Computer Graphics Forum 30*, 3 (2011), 1091–1100. doi:10.1111/j.1467-8659.2011.01958.x. 3

[PNML08a] 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 Transactions on Visualization and Computer Graphics 14*, 3 (2008), 564–575. doi:10.1109/TVCG.2007.70443. 3

[PNML08b] 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 Transactions on Visualization and Computer Graphics 14*, 3 (2008), 564–575. doi:10.1109/TVCG.2007.70443. 7

[PSM14] PENNINGTON J., SOCHER R., MANNING C.: GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (Oct. 2014), pp. 1532–1543. doi:10.3115/v1/D14-1162. 4

[PSN10] PAULOVICH F. V., SILVA C. T., NONATO L. G.: Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1281–1290. doi:10.1109/TVCG.2010.207. 3

[RFaT16] RAUBER P. E., FALCÃO A. X., TELEA A. C.: Visualizing time-dependent data using dynamic t-sne. In *Proceedings of the Eurographics / IEEE VGTC Conference on Visualization: Short Papers* (Goslar, DEU, 2016), EuroVis '16, p. 73–77. 4

[RS00] ROWEIS S. T., SAUL L. K.: Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 5500 (2000), 2323–2326. doi:10.1126/science.290.5500.2323. 3

[SA15] SEDLMAIR M., AUPETIT M.: Data-driven evaluation of visual quality measures. *Computer Graphics Forum 34*, 3 (2015), 201–210. doi:10.1111/cgf.12632. 2

[SH05] SHAO C., HUANG H.: Selection of the optimal parameter value for the isomap algorithm. In *Proceedings of the 4th Mexican International Conference on Advances in Artificial Intelligence* (2005), MICAI'05, p. 396–404. doi:10.1007/11579427_40. 1

[SHK*14] SRIVASTAVA N., HINTON G., KRIZHEVSKY A., SUTSKEVER I., SALAKHUTDINOV R.: Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 56 (2014), 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html. 4

[SLZ12] STUHLSATZ A., LIPPEL J., ZIELKE T.: Feature extraction with deep neural networks by a generalized discriminant analysis. *IEEE Transactions on Neural Networks and Learning Systems 23*, 4 (2012), 596–608. doi:10.1109/TNNLS.2012.2183645. 3

[ST02] SILVA V. D., TENENBAUM J. B.: Global versus local methods in nonlinear dimensionality reduction. In *Proceedings of the 15th International Conference on Neural Information Processing Systems* (2002), NIPS'02, p. 721–728. 6, 7

[SVM14] SORZANO C. O. S., VARGAS J., MONTANO A. P.: A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877. 2

[TBB*10] TATU A., BAK P., BERTINI E., KEIM D., SCHNEIDEWIND J.: Visual quality metrics and human perception: An initial study on 2d projections of large multidimensional data. In *Proceedings of the International Conference on Advanced Visual Interfaces* (2010), p. 49–56. doi:10.1145/1842993.1843002. 2

[TCL*13] TAM G. K., CHENG Z.-Q., LAI Y.-K., LANGBEIN F. C., LIU Y., MARSHALL D., MARTIN R. R., SUN X.-F., ROSIN P. L.: Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics 19*, 7 (2013), 1199–1217. doi:10.1109/TVCG.2012.310. 4

[Uly16] ULYANOV D.: Multicore-tsne. https://github.com/DmitryUlyanov/Multicore-TSNE, 2016. 8

[vdMH08] VAN DER MAATEN L., HINTON G.: Visualizing data using t-SNE. *Journal of Machine Learning Research 9*, 86 (2008), 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html. 1, 3, 4

[vdMPvdH09] VAN DER MAATEN L., POSTMA E. O., VAN DEN HERIK J.: Dimensionality reduction: A comparative review. 2, 7

[VGS*20] VERNIER E. F., GARCIA R., SILVA I. P. D., COMBA J. L. D., TELEA A. C.: Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum 39*, 3 (2020), 241–252. doi:10.1111/cgf.13977. 4

[VK06] VENNA J., KASKI S.: Visualizing gene interaction graphs with local multidimensional scaling. In *Proceedings of the 14th European Symposium on Artificial Neural Networks* (2006), pp. 557–562. 7

[WFC*18] WANG Y., FENG K., CHU X., ZHANG J., FU C.-W., SEDLMAIR M., YU X., CHEN B.: A perception-driven approach to supervised dimensionality reduction for visualization. *IEEE Transactions on Visualization and Computer Graphics 24*, 5 (2018), 1828–1840. doi:10.1109/TVCG.2017.2701829. 2

[WHRS21] WANG Y., HUANG H., RUDIN C., SHAPOSHNIK Y.: Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research 22*, 201 (2021), 1–73. URL: http://jmlr.org/papers/v22/20-1061.html. 1, 6

[WVJ16] WATTENBERG M., VIÉGAS F., JOHNSON I.: How to use t-SNE effectively. *Distill* (2016). doi:10.23915/distill.00002. 1, 3, 6, 10

[XRV17] XIAO H., RASUL K., VOLLGRAF R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. arXiv:1708.07747. 4

[ZGZ*19] ZHU H., GUO B., ZOU K., LI Y., YUEN K.-V., MIHAYLOVA L., LEUNG H.: A review of point set registration: From pairwise registration to groupwise registration. *Sensors 19*, 5 (2019). doi:10.3390/s19051191. 4