Universiteit Utrecht

Utrecht University

Artificial Intelligence

Masters Thesis

Visual Analytics for Improving Deep Learning Multidimensional Projections

---

Author:
Terri Modrakowski
(6498914)

Supervisor:
Prof. dr. Alexandru C. Telea

Second Supervisor:
Dr. Michael Behrisch

External Supervisor:
Mateus Espadoto

July 2020

Abstract

Deep learning has recently shown the ability to construct dimensionality reduction, or projections, with high quality and computational scalability. However, such methods have the major drawback of operating as black boxes, hence, it is hard for users to fine-tune them to achieve more specific projection styles. An important instance of this problem is the learning of t-SNE projections: The learned projections are typically fuzzier than the original t-SNE ones, making them less suitable for many visual analysis use-cases for which t-SNE was originally proposed. We aim to adapt and use classifier visualization methods to get a better understanding of the reasoning behind the network's inference of projections. We pinpoint, and apply fixes to ultimately reduce the causes of diffusion in the learned projections, culminating in the application of KNNP, a nearest-neighbors approach to the original NNP which further increases the quality of deep learning projections.

Acknowledgements

I wish to express a specific sincere appreciation to my supervisor Professor Alexandru Telea, for his patience with me, encouragement and access to his elevated insight.
I want to thank Doctor Michael Behrisch for agreeing to be my second supervisor, being supportive and enthusiastic during demonstrations of my work and for providing assistance when I needed it.
I convey my admiration for Mateus, for his proficiency in deep learning models and skill in simplifying difficult concepts.
I express the deepest gratitude to all of my supervisors for providing boundless inspiration by being themselves, giving me their valuable time and allowing me to experience and be a part of their genius. I have gained so much by the freedom and guidance to experiment and explore within the field of their expertise.
I am indebted to the facilities at the university of Utrecht, which allowed me the exposure to such academic pursuits in my areas of interest. I recognize the invaluable input of my mother, Trish Modrakowski, in her affection for computing and for my ability to be standing here today. And finally, Francesco Vaglienti and Dr. Amelia Beans for bringing stability and comfort to my studies.
To everyone who was a part of my thesis, whether a contributor or bystander, thank you for being a kindred spirit and keeping the passion for artificial intelligence and computing alive.

Contents

Chapter 1

Introduction

Following the evolution of technology, more and more professions are having to deal with larger and larger data. The information contained in this data has become increasingly available and necessary to form a competitive edge. Such data is large in terms of the amount of observations and the features each contains within. Every additional feature adds another dimension of complexity, increasing the difficulty in processing this kind of data. This difficulty comes in the form of large computational time and resource costs. The impossibility of human users assimilating so many dimensions, creates a dependency on this difficult computation. To bridge the gap between large amounts of useful data and human memory limits, dimensionality reduction methods are commonly used. These dimensionality reduction methods aim to create abstractions of the data, abstractions which distill the data into the most digestible interpretation of each sample's discriminatory characteristics.

These abstractions, called *projections*, condense the high-dimensional data into a low dimensional representation while maintaining the underlying structure as much as possible. The structure defines the distribution of the data and how the points relate to each-other, whether in simple groups or in more complex relations. Often this low dimensional mapping is conveyed as a scatter-plot, which allows the user to reason about the original structure by observing the shapes and placements in the projection. Similar observations that exist in the data are grouped into clusters, which allow projections to scale well with regard to the space they require to visualize any number of samples and total dimensions.

1.1 t-SNE Projection

One of the most widely used methods of projection is t-SNE [1]. t-SNE is widely praised for providing good visual separation when placing similar featured points into groups on a scatter-plot. However, t-SNE has quadratic runtime depending on the number of samples and parameter dependent, meaning that the incorrect choice of any such parameter may have a large cost in computational time. t-SNE is non-deterministic, lacking stability and generalization. This means that there is unpredictability and small changes in the data lead to reproductions of the projection which are completely different.

1.2 Deep Learning Projections (NNP)

As an alternative to t-SNE for dimensionality reduction, a deep-learning approach called Neural Network projection (further called NNP) is proposed in [2]. NNP has a compact implementation, making use of a fully-connected, feed-forward *regression* network with relatively few nodes and layers compared to other deep learning techniques. Furthermore this approach overcomes the flaws of t-SNE mentioned above, through its ease of reproduction and out-of-sample capability. NNP trains using any projection technique and a subset of available data to create a mapping from high to low dimensional space. Once the training is complete, the majority of computational cost has been paid. The network can then be used to infer indefinitely, on the same or even similar unseen data. This creates stable projections within a fraction of the time

needed for t-SNE for large amounts of data.
The projections created by NNP have some diffusion or 'fuzziness' when compared to the original projection technique used for training. This diffusion creates some doubt in the separation of clusters. This visually blurs the distinctions that make the groups of points different. The borders of similar points become less clear and we lose some of the efficiency inherent to projections. Figure 1.1 shows this diffusion and the lack of clarity in clusters between t-SNE and NNP.

Possible hypothetical causes of this diffusion include: [3]

- Underfitting: either by training too little or by not providing a large enough sample size to be representative
- Overfitting: by fitting too closely to the data through lacking regularization or validation.
- Optimization: by stopping the training before the model reaches its full potential, in cases where the optimizer gets stuck at local optimum instead of finding the best possible solution.

However, since the hyperparameters such as optimizers and regularization have already been studied in [3], we build on this work and try pinpoint the exact cause of diffusion outside of that search space.
This brings us from the parameter choice to the examination of the model itself. Examining the model is non-trivial, as deep learning methods often function as a black box. In such a case, the root-causes of the problems remain hidden which exaggerates the difficulty in understanding the actual issue. Deep learning models are notoriously difficult to interpret, because they are usually very large and use many parameters which lead to decisions that are difficult for humans to trace. In most applications of deep learning the user supplies input and is only aware of the resulting output which is taken at face value, while the internal computations are obscured within. This restricts the users options and forces them to trust the model, tweak it blindly or discard it completely. Due to this and the rapid adoption of deep learning models, there is a surge of interest and need for techniques that inspire trust in these models and help us understand their behavior, especially since their proficiency seems to be a promising direction for advancing the way we look at the information at our disposal. The transparency of this model and the improvement of the diffusion within must be addressed together, as they are compounding problems.

1.3 Research Question

So far we have discussed t-SNE, NNP and the features of both. What is clear is that NNP is strong competition to t-SNE, held back slightly by its diffusion and black-box properties but boasting many powerful advantages. Given some improvements, NNP is not only a strong competitor but also a potential replacement for t-SNE.

How then, can we *improve the quality of deep learned multidimensional projections?* To address this complicated problem we propose the exploration of this deep learning projection technique, NNP, through the adaption and creation of tools within a visual analytics toolkit. With such visual applications we can convey the quality of the projection through quality metrics in order to measure the performance of the model and define improvement and add explanability to the model.

Exploration of the model with such tools will also allow us to get a better understanding of the fundamental behavior of the underlying model, revealing what drives its decision process during projection. Using these tools we gather knowledge of the decision process and gain insight into the creation of the diffusion found in the projections. Once causes of this diffusion are found, we aim to isolate them and therefore focus our search space onto these issues. We then propose and investigate solutions to correct these underlying causes in order to clarify the clusters in the projections. Once the cluster separation is improved, NNP will be much more competitive with industry leaders such as t-SNE. This thesis will detail an exploration using this visual analytics toolbox in an attempt to unearth causes which can be fixed to improve the diffusion in NNP.

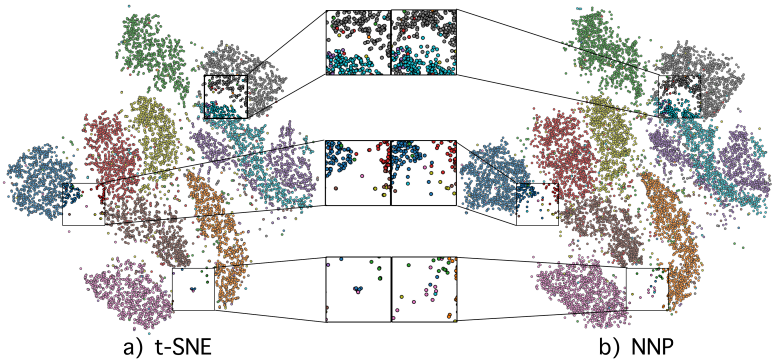


Figure 1.1: Example of diffusion in t-SNE (a) versus NNP (b). Insets show diffusion details.

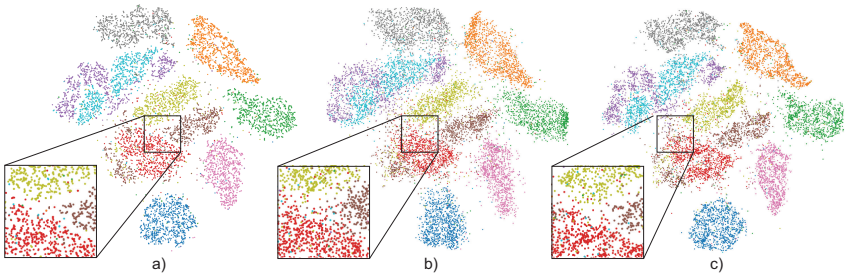


Figure 1.2: Example of diffusion introduced by NNP. (a) Ground-truth t-SNE projection (b) Inferred NNP (10K samples), (c) Inferred KNNP (10K samples). Insets show diffusion details.

1.3.1 Structure of this Thesis

The structure of this thesis is as follows.
In Sec. 2 we examine related work on dimensionality reduction, deep learning and visualization tools. We do this to define the lineage of our work, the terminology, main concepts and the academic context to this thesis.

This thesis is then divided into two parts, one for each sub-question relating to our research question.
The first sub-question is *How can we understand the diffusion or limited quality in projection?* This will be addressed in Sec. 3.2. The second sub-question, *How can we improve the projections once we have this understanding?* will be addressed Sec. 3.7.7. Sec. 4 details the final fix we apply as a result of the findings in Sec. 3, by using KNNP, a variant of NNP which uses a *neighborhood* approach. KNNP improves the visual separation in the projections, as shown in Fig. 1.2.

Finally, in Sec. 5 we conclude our work and suggest potential future research directions.

Chapter 2

Related Work

So far we have briefly discussed t-SNE and its deep learning rival NNP. In this chapter we will go more into detail about these methods as the foundation of this thesis. First in Sec. 2.1, we will introduce some of the core terms related to this work. We then discuss the technical fundamental methods t-SNE (Sec. 2.2) and NNP (Sec. 2.3) in more depth before finishing the chapter with other related work in Sec. 2.4.

2.1 Terminology

2.1.1 Projections

Let $D = \{x_i\}$ be a dataset of N samples so that $1 \leq i \leq N$, where each $x = (x^1, ..., x^n)$, $x^i \in \mathbb{R}$, and $1 \leq i \leq n$ is an n-dimensional sample. Therefore D has N samples or rows, and D has n dimensions or columns.

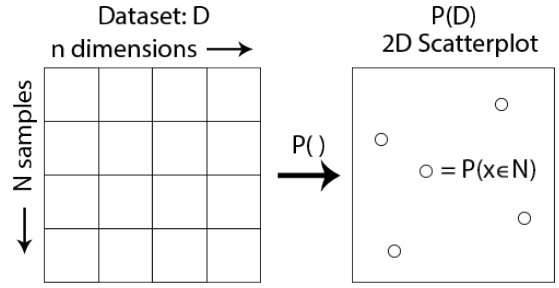


Figure 2.1: The projection from $\mathbb{R}^n$ data to an $\mathbb{R}^2$ scatter-plot

Dimensionality reduction can be used to preserve the high-dimensional structure of D while distilling it into a more interpretable and usable format. This trade-off of dimensionality and interpretability comes at minimizing an objective function that measures discrepancy between the original data D and any resulting format that reduces the dimensions. One method to perform dimensionality reduction is through projections. A projection method is a function

$$P : \mathbb{R}^n \to \mathbb{R}^m \tag{2.1}$$

where $m \ll n$. This maps the same information in each sample to lesser dimensions, generally m = 2. In this way every $x \in D$ is projected from an n-dimensional sample, to a 2-dimensional point $P(x)$ by the projection function $P()$, each of these points are then grouped into a scatter-plot which is denoted as $P(D)$. Projections are particularly valuable in cases where individual identity is less important, [4] which is useful for the goal of creating clusters of points that have a group identity without the fine-grained details.

Many projection techniques exist; we now discuss some examples of projections that are particularly relevant to this thesis. For a more in-depth overview of state of the art projection algorithms a reader may examine [4].

Some projections aim to preserve the distance in the global data, maintaining the overall structure when projecting. The examples we will mention include: PCA, LSP and MDS.

Principal component analysis (PCA) [5] is a technique that creates and projects principal components which maximize variance through solving an eigenvalue problem.

Least Square Projection (LSP) [6] is a method of preserving global structure by approximating the

least squares to the coordinates of a few control points with defined geometry.

Metric Multidimensional Scaling (MDS) [7] is a nonlinear technique that creates a projection based on a matrix of the distances between them. Other projection techniques aim to preserve local structure instead of these global distances, examples of this include: t-SNE (described in Sec. 2.2), LAMP, Isomap and IMAP.

Local Affine Multidimensional Projections (LAMP) [8] is a spring embedding technique that relies on a mathematical formulation derived from orthogonal mapping theory to build local transformations.

Isometric feature mapping (Isomap) [9] is a nonlinear learning technique which creates a low dimensional embedding by computing the geodesic distances between neighbors in the high dimensional space.

Uniform Manifold Approximation and Projection (UMAP) [10] which, based on assumptions in the data, models a fuzzy topological manifold and then minimizes the discrepancy between this and the fuzzy topological structure of a projection.

2.1.2 Metrics

In order to focus our search to improve the diffusion of the points, we require a way to determine where the errors are. Usually when dealing with deep learning methods, a metric of error is defined as the difference between true and predicted values. This is still the case here, as we minimize the training loss to create the NNP model. However, this accuracy is very little indication of how useful the model is as a projection compared to t-SNE. This loss is designed to fit to t-SNE's ground truth coordinates, not a measure of projection performance. For example we could train NNP on t-SNE until NNP is fit with exact precision, but we have gained nothing from this new projection. At best, this model is the same as the t-SNE projection used to fit it. With a model which fits so closely to the t-SNE we likely lose any generalizability. So then how can we measure how well the projection is doing?

To accomplish this we use three methods that are shared by [2]. Both [2] and this thesis evaluate the NNP method, so it makes sense to use these same methods as a baseline for comparison. These methods are in the form of quality metrics, namely: trustworthiness, continuity and neighborhood hit. In [2], these methods are computed at a global scale while ours are not. The average of the metric over the projection does not help us to find small differences. We instead compute these metrics locally per point, to preserve finer local behavior. This is especially interesting as different projection methods preserve small distances better than others [4]. Since we are comparing at least two different projection methods (NNP and t-SNE), it is worth computing these metrics at a finer scale to determine if there is some change in small distance preservation between them. This is not evident since these two methods optimise different kinds of objective functions. NNP is concerned with preserving distances while t-SNE is concerned with preserving neighborhood structure.

Trustworthiness T:

Trustworthiness penalizes the missing neighbors in the neighborhood of k-points of each projected point. Missing neighbors are points which are close in D but not in the projection. Neighbors in D that remain close in $P(D)$ give a good trustworthiness score of 1. For a user this score implies that the neighbors in the data are visually projected nearby and that their appearance replicates the local patterns in D. If we have points with low trustworthiness, *i.e.* points which are close in the projection, but are not supposed to be because the points in D are not close, this will cause the user to misinterpret how the similarity of the points changes with respect to distance.

Given N nodes in the data, and the ranking of nodes $r(i,j)$ as the ranking of sample j according to its shortest distance to the sample at index i and the neighborhood of size k around sample i denoted $U_k(i)$ in the n-dimensional data, Trustworthiness is then:

$$T = 1 - \frac{2}{Nk(2N - 3k - 1)} \sum_{i=1}^{N} \sum_{j \in U_k(i)} (r(i,j) - k) \tag{2.2}$$

Continuity C:

Continuity measures false neighbors in the neighborhood of k-points of each projected point. This

penalizes points that are close in the projection but not in D. Neighbors in $P(D)$ that are consistently close in D give a good continuity score of 1. For a user this score implies that points that the user sees nearby in the projection are actually nearby. If we have points with low continuity, *i.e.* points which are close in D but are not close in the projection, this will cause the user to see points as much more similar than they actually are by misjudging how far away significantly different points are.

Given N nodes in the projection $P(D)$, and the ranking of nodes $\hat{r}(i,j)$ is the ranking of point j according to its shortest Euclidean distance to point i in $P(D)$ and the neighborhood of size k around point i is denoted $V_k(i)$ in the projection,
Continuity is then:

$$C = 1 - \frac{2}{Nk(2N - 3k - 1)} \sum_{i=1}^{N} \sum_{j \in V_k(i)} (\hat{r}(i,j) - k) \tag{2.3}$$

Neighborhood Hit NH:

Neighborhood hit compares the neighborhood for homogeneity in class label. This measures the separation of clusters, penalizing by neighborhoods with different class labels. A high score of 1 occurs where all of the points in the neighborhood have the same label, in a perfect separation from other points. This requires that the clusters we want to see completely separated actually exist in the data, and that we have class-labels for computation of the metric.

Note that we do not use labels in the computation of either t-SNE or NNP, so the choice of a metric that evaluates labels may seem strange, but generally the label is a good indicator of similarity. More importantly, this similarity allows us to measure neighborhood preservation which is prioritized by t-SNE. Prioritizing this metric is then a good indicator for success in mimicking t-SNE.

For a user this score implies that a point is with others of the same kind and far enough away from other points.

We denote the neighborhood k of point $y \in P(D)$ as y_k and the number of shared labels as y_k^l. Neighborhood hit is then:

$$NH = \frac{1}{N} \sum_{y \in P(D)} \frac{y_k^l}{y_k} \tag{2.4}$$

Shepard Diagram [8]:

A Shepard diagram is a scatter plot which compares how far apart data points are pairwise in $P(D)$ and D. The main diagonal line indicates how well preserved the distances are when projected.

Spearman Rank Correlation R:

R can be used to measure the Shepard diagram correlation through its quantitative measurement of relations between two variables. For a user, a high R value means that the distances in the input are well preserved in the output projection and that the projection fits well to the data.

Normalized Stress [11]:

Normalized Stress also measures the differences in the projection and data, creating a global badness-of-fit measure. This is a *badness measure* instead of a quality metric, so it is inverted. A user would like to see a score of 0, which indicates a lossless mapping, between D and $P(D)$. This lossless mapping would mean that no information is lost in the compression of the n-dimensional data during the projection process.

The R metric, which is present in previous iterations of work on NNP, was exchanged in favor of such a stress metric in the proposed visualization. However, this was omitted from the final solution as it was found to be a poor indicator of diffusion, as shown in Figure 2.2. In this figure the lighter, more yellow points score highly, while the darker, more red points have high error. We see that the points that are diffuse suffer less from stress, while the points within the clusters that we do not consider diffuse are penalized heavily. Furthermore, there is no clear separation in color of these two groups, as not all diffuse points are lightly colored and not all internal clusters are dark.

2.1.3 Deep Learning

Also particularly relevant for this work is the notion of artificial neural networks (ANN). A neural network is an algorithmic model that can learn patterns. These are made of artificial neurons placed into layers which are linked together. A *fully connected* network has each neuron in a layer connected to every neuron in the next layer.

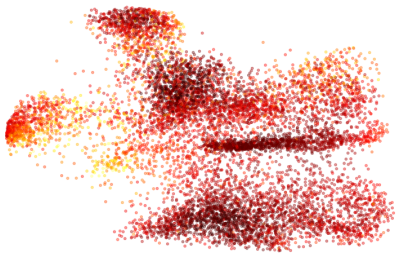


Figure 2.2: A heat colormap of the stress metric on an NNP projection

Each neuron computes a linear function with added bias using the output of the neurons in a previous layer.
This is in the form:

$$z_j^{(i)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} a_k^{(l-1)} \tag{2.5}$$

where $b_j^{(l)}, w_{j,k}^{(l)} \in \mathbb{R}$ are the bias and weights respectively, and $a_k^{(l-1)}$ is the activation of a connected neuron k in the previous layer $l-1$. Each neuron computes a linear combination of the previous activations weighted before adding a bias.

The two most influential components within a neural network are the loss function and activation function. The loss function helps the network to learn what is correct. The activation function allows the network to approximate a wider range of mathematical functions. In the next sections we will discuss these components and the specific variations chosen for this thesis.

Loss Functions

The learning in a network occurs as a result of weight adaption according to the minimization of a loss function. This adaption is computed through back-propagation of error, which commonly uses a method such as gradient descent. Gradient descent is used to compute the direction needed to tune the sensitivity of weights through the network. Once tuned these weights minimize the error across the entire network and every training sample. In the original NNP paper [2], MSE (mean squared error) was used but during hyperparameter tuning [3] MAE (mean absolute error) performed better.

Since we build on that model, in this case we use the loss function MAE:

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{2.6}$$

where y_i is the expected true value, and $\hat{y}_i$ is the predicted answer to the same sample as given by the network. In this way we minimize the difference between expected and predicted to guide the network towards correct choices.

Activation Functions

Recall that the neural network function is just a computation of a previous activation, multiplied by a weight and a bias, so this is a linear equation. If we were to try to fit a linear equation to the hidden function within the network, we would lose a lot of important information. To help the network more easily approximate a wider variety of more complex non-linear functions, the activation function adds a non-linearity. Activation functions are applied to each of these linear functions, and passed onto the next layer for this process to happen again. Layers subsequent to the input layer are called hidden layers. These hidden layers make up the black box that obscures the computation of the Neural Network. In this case, we activate our hidden layers using a rectified linear layer, which removes negative values replacing them with 0, this is commonly called ReLU:

$$a_j^{(l)} = max(0, z_j^{(i)}) \tag{2.7}$$

The final layer or output layer is where the result of total computation is revealed. For projection purposes we require a pair of coordinates between 0 and 1. For this final layer we use a 2 neuron sigmoid activation of the form:

$$a_j^{(l)} = 1/(1 + exp(-z_j^{(i)})) \tag{2.8}$$

Activations as Learning

The result of a neural network can come in the form of a prediction of a discrete (classification) or continuous (regression) value. Once created the model can be used repeatedly for inference. Inference involves making predictions for any data similar to training samples at a fraction of the computational

cost of initial training. Activation values change according to the weighted features in the input to minimize loss. This means that these features measure how much a node contributes to a certain kind of point. Thus, activations can be interpreted as an abstraction of the given observation. These abstractions reveal which features of the input the network has learned and the decisions the network has made to differentiate the samples.

2.2 t-SNE

As mentioned in Sec. 1 the projection technique we wish to emulate is t-SNE [1]. t-SNE is a non-linear dimensionality reduction technique, part of the family of SNE (*Stochastic Neighbor Embedding*) techniques. It is currently very popular for its high degree of neighborhood separation and is considered a gold standard in 2D Projection.
t-SNE uses *Stochastic Neighbor Embedding* by constructing a probability distribution over pairs of objects in D. This distribution is such that similar objects are assigned a higher probability while dissimilar points are assigned a very low probability. This probability is computed by focusing a Gaussian distribution over a point in a neighborhood and noting the probability distribution at that point. There is a scaling factor to account for density in points and an averaging operation where the points meet in each other's distribution.
Then t-SNE defines another distribution over the points in $P(D)$, except this time it uses a t-distribution instead. The goal is to make the probabilities in D the same as $P(D)$; to do this the similarities must be measured. The measurement of similarities across these two probability distributions is taken using the Kullback–Leibler divergence between them with respect to the locations of points, which is minimized by gradient descent. This divergence metric penalizes differences specifically when similarity is large in D. This penalization helps to preserve the local structure while not emphasising further differences as much. The heavier 'tail' of the t-distribution allows additional compromise when placing points further than they would accurately be, which paired with disregard to the placement of points that are not at a neighborhood scale by the divergence metric, leaves t-SNE with large cluster separation that is accurate at a local scale but may exaggerate gaps between neighborhoods. The t-SNE projection iteratively fits its points by an accumulation of the attractive forces of similar points and the repulsive forces of dissimilar points

2.2.1 Benefits of t-SNE

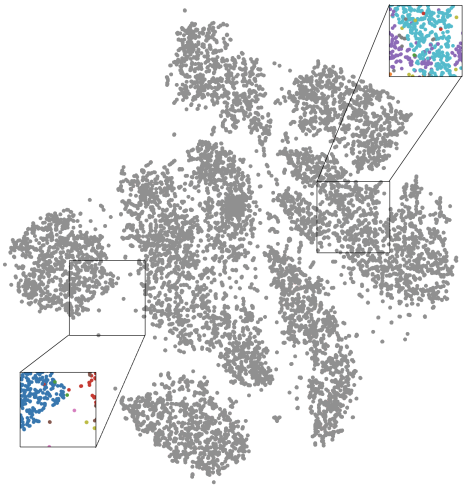


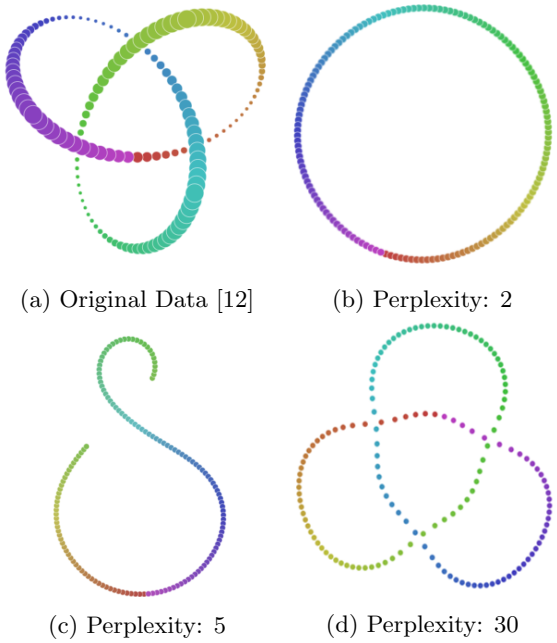
Figure 2.3: Cluster separation without class labels, well separated (left) and indistinguishable (right). Insets provide class information.

The separation of neighborhoods clearly in visual space is a substantial benefit of using t-SNE. Consider the case of unsupervised learning where no class data is available, such as the case of Figure 2.3 where we have omitted the class data from the main projection. In such a case we are only able to tell that there are distinct groups because t-SNE provides clear visual separation between groups as seen in Figure 2.3 (left). In the right of Figure 2.3 we see the opposing side; two clusters which are not well-separated and the user has no way of knowing they are two mostly distinct groups. These two groups happen to overlap in similarity, so even t-SNE struggles to split them clearly, but without the visual separation prioritized by t-SNE, all groups may be similarly indistinguishable.

2.2.2 Disadvantages of t-SNE

Parameter Sensitivity

The benefit of cluster separation, as mentioned in Sec. 2.2.1 can be tricky to achieve as t-SNE is very sensitive to parameterization. This sensitivity makes implementation complicated, as some domain expertise may be required to convince it to behave as intended.



(a) Original Data [12] (b) Perplexity: 2

(c) Perplexity: 5 (d) Perplexity: 30

Figure 2.4: How t-SNE varies with respect to parameters, courtesy of [12]

In Figure 2.4 we demonstrate the range by which t-SNE fluctuates based on a single parameter – perplexity. Each projection generated by t-SNE tells a very different story about the structure of the data and may mislead a user.
A search through the space of potential parameter settings would take a large amount of time. Moreover, even once the optimal parameter values were found, they would not generalize to other datasets or even data subsets of different densities. Hence, t-SNE is hard to use as a *replicable* method.

Quadratic Nature

t-SNE has a quadratic runtime, subject to the number of points, making it very slow for large datasets and completely infeasible on very large datasets. This becomes especially problematic with the other drawbacks of t-SNE. Specifically, that the initial dataset must contain all observations of interest because of t-SNE lacking consistency and out of sample support.

Out-of-sample

t-SNE cannot project out-of-sample data, which means that every point of data that needs to be projected in an experiment space needs to be present at the creation of the model and any new additions require a full re-computation. These re-computations come with some other issues, which are outlined in this section.

Consistency

Consistency is a complication that arises with recomputing t-SNE. If a user wishes to add new data or simply another subset of the same data they must recreate a projection. Unfortunately t-SNE has no stability and is stochastic. Models change disproportionately with respect to any alterations. This randomness makes subsequent projections on similar data inconsistent and prevents a user from making comparisons between two projections created on very similar data.

2.3 NNP

The projection technique we wish to improve is NNP [2]. NNP is a Deep learning projection technique, which means that it is an ANN that learns without class-labels. NNP takes D and some projection already computed on D, $P(D)$ as input. D can be any kind of high-dimensional data that can be represented as multidimensional vectors, such as image or sentiment datasets.

2.3.1 Architecture of NNP

A compatible input is fed into a shallow, fully-connected, regression neural network which learns how to place D with respect to the 'truth' in $P(D)$ by minimizing the MAE (EQN. 2.6) between actual t-SNE $P(D)$ values and NNP predicted $P(D)$ values.

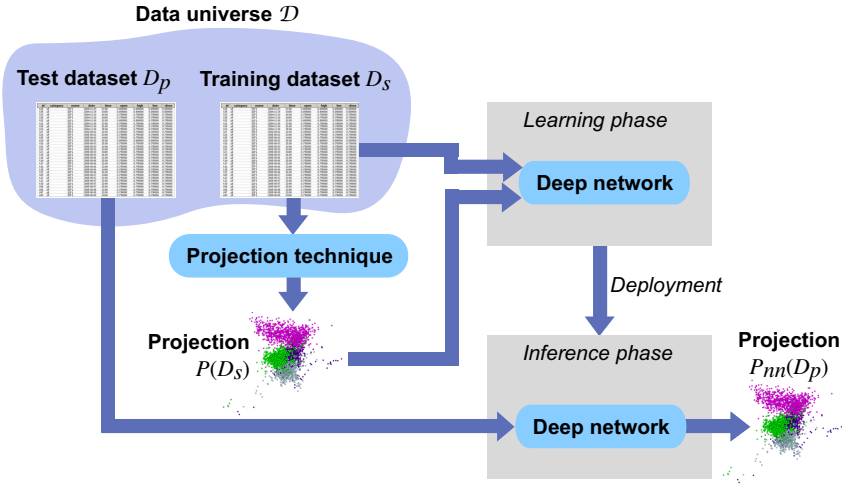


Figure 2.5: The learning process of NNP (Figure from [2])

The hidden layers of the network activate using the ReLU function described in EQN. 2.7 and the output layer returns a 2d coordinate as a result of a sigmoid function, outlined in EQN. 2.8. The decision of these components follows the parameter optimization in [3]. The layer sizes in Figure 2.6 correspond to a shape and style of standard/wide, where standard refers to the multiplier of nodes, while wide refers to the pattern given to a layout of nodes. Regardless of layout, NNP has a small amount of layers which makes it easy to implement.
Other layouts are listed below:

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

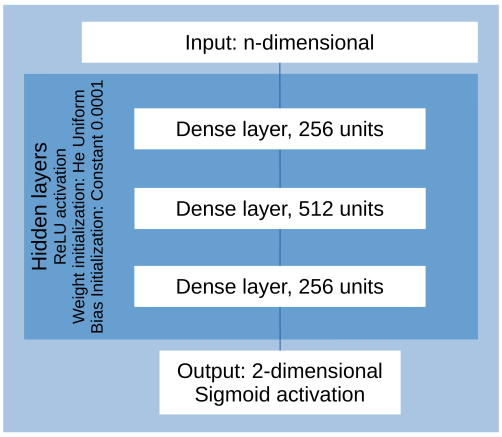


Figure 2.6: The architecture of NNP, showing the hidden layers and activation functions of each. (Figure from [2])

2.3.2 Benefits of NNP

We have positioned NNP as a strong competitor to t-SNE, this is because NNP projects very similarly in terms of performance, while excelling exactly where t-SNE falls short as listed in Sec. 2.2.2. NNP is incredibly simple to use and parameter agnostic as shown in [3], as opposed to t-SNE which heavily requires expertise to set its parameters. NNP is much faster than t-SNE and is stable with regards to small changes in the data [2].

Parameter Sensitivity

NNP is stable with regard to parameters, as shown in [3]. This was proven by tuning the training-set size, node quantity, layer shape and selecting the best performing regularizers, optimizers and loss functions. The end result was a slight improvement in results from the original NNP and a verification that the NNP method does not heavily rely on any of these tune-able parameters to perform well. This optimization did not specifically target the diffusion, but we will build off of the work in [3] to continue to search the solution space in a direction orthogonal to hyper-parameter tuning.

Linear Complexity

To make the most effective use of training time, NNP uses 'early stopping', which is effectively a heuristic to determine when to stop training.
The loss is monitored and if it does not go down by at least δ in the number of epochs given by the patience parameter, the model assumes this is the maximum and stops training. This has the potential drawback of getting stuck at a local optimum but often the training investment for a true maximum is not worth the performance difference from a local maximum. This training investment time is linear with respect to number of dimensions and observations, regardless of the technique NNP learns from.

Out-of-sample

Due to the ease at which NNP recognizes the structure in the training projection, it generates projections which are difficult to distinguish from the original training projection at a glance. This structure is present even when NNP is projecting data that was not part of the training set, *i.e.* unseen test data, therefore content that does not actually exist in the learned projection will have similar structure.

Consistency

The stability of NNP means it will project the same or a similar point in approximately the same place once trained, as it learns where to place based on abstractions of the features within the data. By virtue of being a Neural network, NNP is able to create stable projections on unseen data. Neural networks require only initial training, at which point most computational resources in time and processing are invested. The model is then reused on similar data for inference multiple times, assuming that the training data is somewhat representative of the unseen data.

2.3.3 Disadvantages of NNP

The out-of-sample capability, stability of NNP and speed comes at the cost of the diffusion of some points, which can be seen as a reduction in cluster-separation and in quality metrics. This is a large issue, as the popularity of t-SNE is due to its cluster separation which NNP loses through diffusion.

2.4 Visual Exploration

In this chapter we describe the literature which provides context to this thesis. Specifically we describe similar methods of visualization to those we are going to use. Both to present our projections, but also to understand how certain visualization methods work and gain inspiration from these. Next we give a necessarily brief overview of such visualization methods.

Similar in purpose to our proposed solution, in [13] a pipeline is used to explore, visualize and refine machine learning. Although this toolkit is much larger in scope than ours, it provides a good example of the kind of iterative, interactive and interpretable framework for AI methods which we aspire to create.

2.4.1 Visualization of Classifier Decisions through Activations

A classifier is a kind of neural network that are trained to predict discrete class labels. Classifiers make clear cut decisions by grouping samples into one of a subset of categories. Perhaps because of this strict decision process they are very popular for exploration with visualization.

Discriminatory Neurons in Classification

The representations of observations learned by the neural network on three image classification

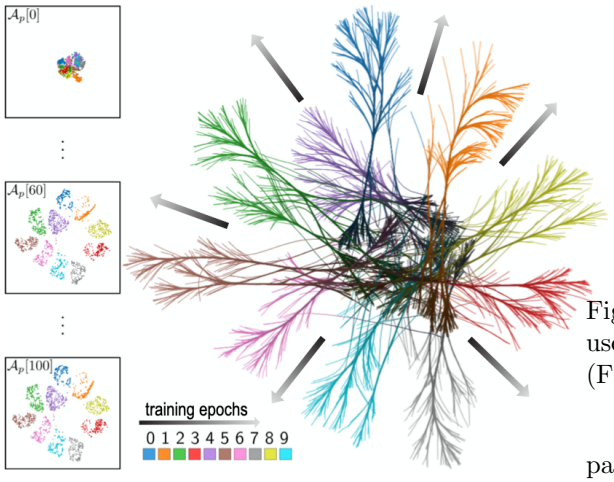


Figure 2.7: A visualization showing the evolution of cluster separation in a classifier over progressive training epochs. (Figure from [14])

datasets are visualized in [14]. The understanding of the evolution of differing representations and inter-neuron relations is used to plot a trajectory of the network's learning during training. By scoring the contribution of neurons, they successfully isolate certain discriminatory groups of neurons which contribute the most to certain class labels. Similarly in [15] the activations of the neurons are used to distinguish misclassifications from the correctly classified points. These approaches are able to show how the contribution of neurons can be used to isolate discriminatory groups to reveal the decision process.

Discriminatory Super-pixels in Image Classification

Similar to these previous classification visualizations, the paper [16] introduces the LIME technique which aims to explain predictions through visual representations that show the relationship between observations and the predicted outcome. This brings specific emphasis to interpretability in models, accounting for the user's limitations when designing visualizations. In particular they show a selection of super-pixels on top of an image classification sample to convey easily to a human which

(a) Husky classified as wolf (b) Explanation

Figure 2.8: LIME visualization of super-pixels, used as a discriminatory measure for explanations. (Figure from [16])

parts of an image were responsible for a class prediction. These super-pixels are a form of local explanation which approximates the closest decision boundary. This explanation contains the discriminatory information which guides the decision process of the network. This decision in visual form, can be conveyed to and interpreted by a user.

2.4.2 Other Learned Representations

Broader methods in the visualization of deep learning concern models which are not as simple as the NNP model. These more complex models learn in different ways which give them more options to convey their decision process.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a form of temporally sensitive ANN, where each time step has a recursive state. The hidden representations of these intermittent recursive states are obscured. In LSTMVIS [17] these hidden states are visualized and the technique is able to use this to return similar learned representations to a user supplied hypothesis.

Convolution Neural Networks

Another commonly used visualization application is that of convolutional neural networks (CNNs). These are networks which slide filters over an image and activate where these filters match; matching defines a feature of the image.

These features can then be visualized to intuitively see what that filter specializes in, for example a filter may learns to detect edges in an image. In [18] the activations of user input are displayed in real-time across the feature maps in the convolution layer and traced upwards into the output layer to show the user exactly how the network formed a prediction.

2.4.3 Visual Analytics

Regarding principle of user interaction within visualization tools for analytics, papers such as Disfunction [19] and FDive [20] allow the user to shape the model interactively.

Dis-function proposes a method to adapt the distance function through users dragging a point in a scatter-plot to facilitate an update of feature weights.

FDive uses relevance models and pattern-based similarity to rank features by how they distinguish their data. A user can then navigate multi-dimensional data by changing what is or is not considered relevant data by labelling a point.
The relevance is updated and FDive will emphasise the discriminatory features on similarly relevant data.

In both cases the user is guided into reaching the results that are relevant to them by directly shaping the feedback that they receive.

2.4.4 Distinctions from this Thesis

Distinctions in Visual Analytics Techniques

There are two distinctions between the previously mentioned methods and our methods which we present in Sec.3 for visualizing NNP.
Firstly, most listed methods are focused on explaining classifiers or some categorical quality and NNP is a regression network. Instead of predicting a class label or category we predict coordinate placement, so we cannot declare something a misclassification. We can only compute how far it is from ideal and we do not have the liberty of having limited discrete outcomes to predict. Secondly, certain models like convolutional neural networks have feature maps, or some other view-able learned representation. Our model is a simple feed forward network which has no such abstraction component. Due to these two distinctions the application of visualizations to NNP is seemingly novel.

Distinctions in User Interaction

Recall that FDIVE and Dis-function allow the user to shape the feedback they receive. These methods demonstrate the importance of user input, in our method the shaping of user feedback is mostly constrained to the on the fly re-computation of distance metrics according to new neighborhoods, or in the usage of specific tools. This is because instead of relying on a user to shape our model, requiring their input to reduce error and possible obscure causes, we would rather use our quality metrics as an objective measure of the projection's performance to guide exploration into explicit causes.

2.4.5 Similarities with this Thesis

The most relevant part to our NNP problem is the ability to isolate discriminatory groups in order to interpret the network's decision process.

In 'Visual Analysis of Dimensionality Reduction Quality for Parameterized Projections' [21], Martins et al. allow the user to access a set of interactive visualizations which help iteratively explore the space of parameter choices.
The user has the ability to directly select points of interest in the visualization for scrutiny by the quality metrics. The user has access to freely available perspectives, and is able to isolate and focus on selected problematic points. Our method accomplishes many of the same things - allowing multiple perspectives, isolation methods and focus areas - but instead of computing the metrics based on the selection of a point, we compute metrics as individual values with respect to a neighborhood. We believe such a neighborhood is the unit of value when mimicking a projection with such specific neighborhood preservation as t-SNE.

We also emphasize the great importance of the concept of activations as learned representations as being fundamental for this thesis, as proposed in papers such as [21].

Chapter 3

Visual Exploration of Deep Learned Projections

In the previous chapter we introduced the background of our research, and what we hope to achieve in contrast to other approaches.
The first part of our research goal is: *How can we understand the diffusion or limited quality in projection?* To get this understanding we use the metrics we outlined in Sec. 2.1.2. We apply these metrics as the first of our tools, in order to discriminate between the well and badly performing points. The arrangement of these points outlines the hot spots of diffusion.

3.1 Examining the problem: Applying our Tools

The goal of our research is to be able to explore the black box of the neural network. By designing and applying tools to NNP within our solution, we aim to visualize and explain the reasoning behind NNP's choices. We can then use this understanding to find areas which show potential for improvement. We will now detail our visualization solution, starting with the general workflow and the interface.
In Figure 3.1 we outline an overview of our method, the starting steps are the same as those in Figure 2.5, with the additional workflow of this thesis. Examining the problem begins with a heat colormap applied to each projection, as a basis for all experiments. We use the heat colormap in each view to locate the error. $Pnn(D_s)$ and $Pnn(D_p)$ are the projections created by NNP that we wish to improve, so we focus on the error in those views in particular.

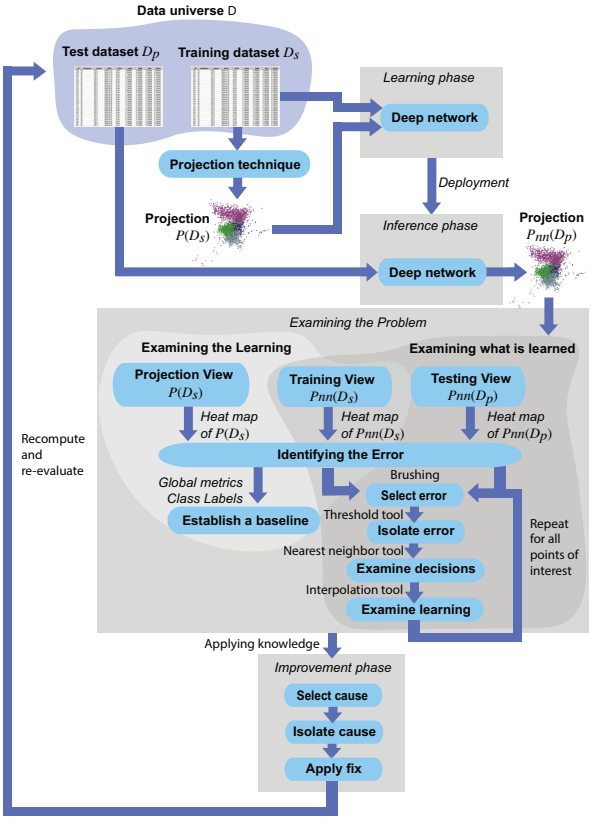


Figure 3.1: An overview of the exploration process in this thesis

When hot-spots of diffusion are found, we start the cycle of selecting, isolating and exploring this error. We repeat this process for a subset of interesting points to gain insight into the causes behind

the error and outline possible causes. Once we have documented causes we start the process of selecting and isolating again but this time with the noted potential causes instead of error. Once the causes are isolated we can apply fixes. Finally, we recompute and evaluate the improvement, continuing from the best performing model at that time.

Initialization

In the following section we introduce the datasets and the available parameters for initialization of NNP within the toolbox.

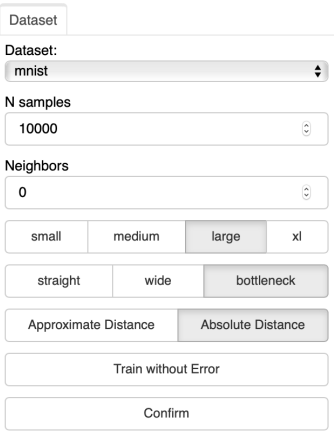


Figure 3.2: The first screen of the proposed visualization solution

NNP Parameters
When the user first enters our proposed visualization solution, they will be met with the initialization screen. This is for the parameters to train the model, which are required before any exploration can take place. The N samples input allows the user to select how many samples they want to include in the training and test set, the sizes and styles are discussed in Sec. 2.3.1. 'Train without error' is related to removing points in Sec. 3.6.2. The option for neighbors has to do with the implementation in Sec. 4. The distance choice allows the user to opt for a time-accuracy trade-off: the visualization solution becomes faster to start up by selecting an approximate distance [22] that may be less accurate instead of absolute (Euclidean) distance.

Finally the confirm button signals to the visualization solution that the user has selected the parameters that they would like, and begins to generate the model.

Unfamiliar Options on the UI
A reader may notice the exclusion of standard/wide option for size/style. This is because the size conventions of that architecture are not as easy to generalize into other styles while remaining consistent with the existing styles. The model using standard/wide itself did not stand out in any particular case. Furthermore an 'xl' option was added, particularly for exploration with the architecture introduced in Sec. 4. The xl architecture is:

- *Xl - straight*: 960, 960 and 960 units;
- *Xl - wide*: 720, 1440 and 720 units;
- *Xl - bottleneck*: 1200, 480 and 1200;

We do not commonly use this architecture in these experiments, particularly because it did not bring additional results and did have an increase in computation time. It is important to mention this additional architecture, because a fear of KNNP in Sec. 4 is the network size throttling the input and by using this size we made sure to provide KNNP enough space.

Datasets:
A user can select from a list of currently supported datasets, which are a subset of those used in the original NNP evaluation [2]: **MNIST[23]** and **Dogs *vs* Cats [24]**. Any dataset that works with NNP is compatible, however some datatsets will need an adaption to specify the things they want to visualize. For example, a text database would need slight adaptions to show a tokenized word instead of an image in the hover tool.
MNIST [23]: 70K observations of handwritten digits from 0 to 9, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors;
Dogs *vs* Cats [24]: 25K images of varying sizes divided into two classes (cats, dogs). We used the Inception VGG16 [25] convolutional neural network pre-trained on the ImageNet data set [26] to extract features of those images, yielding 2048-element vectors for each image.

While the vectors are used for the model itself, the raw images are used for the exploration of the resulting projection in the hover tool as a means to convey trust to the user.

(a) Dogs vs. cats: 2 classes, 25k samples [24](Figure from [2])

(b) MNIST: 10 classes, 70k samples [23](Figure from [2])

Once the user has clicked confirm, the projection of t-SNE, NNP training and NNP test views are computed. These are the three models/views that form the basis of the examining the problem in the workflow Figure 3.1.

Caching

The models created on the back-end are saved using a unique identifier flag according to the dataset, parameter set and state of the model, so the same parameters on multiple runs will resume an older model. This makes subsequent runs much faster, as our solution has significant computational overhead from the original NNP. We compute t-SNE as a training projection and NNP, which are both part of the basic process of NNP, but for the visualization we need to compute activations for every layer, as well as distance computations between training, test and ground-truth projection. This is not a desired process, but a coping mechanism to allow the proposed visualization solution to be interactive. Likely an optimization could easily overcome this problem but that is not within the scope of this thesis.

3.1.1 Projection, Training and Testing Views

In this section we only detail the basics of tools as not to overwhelm the reader; we go into further detail of tools as they become relevant for the experiments we perform in Section 3.3.2.
All components of our visualization solution fit into 'views' divided into dataset types for ease of use and specialization within the space of examining of the problem. (Figure 3.1):

- Projection View: Examine global metrics to examine the ground truth.
- Training View: Examine the learned structure and comparison to ground truth.
- Testing View: Examine the inferred structure and generalizability, relations between neurons, which features are learned and the effect of the training process.

The user is able to use any appropriate tool in any view, as many of the tools listed in previous views are available in others. For example, the test view contains all tools within the projection and train view. However this inheritance is order dependent because neurons do not exist in the projection, so the tool for examining neuron relations is unavailable in that view. The above list is the intended usage for our experiments but the user is in no way restricted.

In this section we begin to examine how the input affects the diffusion through instance based interpretations of training sample neighborhoods. In this way we introduce the user to a baseline example of our proposed solution that forms the first steps of our evaluation.

Figure 3.4 shows the *Projection View*, which shows the ground truth projection that NNP will use as the 'correct' predictions for the training set. This is our least detailed view, and as such the other views inherit from it, *i.e.* every other view has these tools and more.

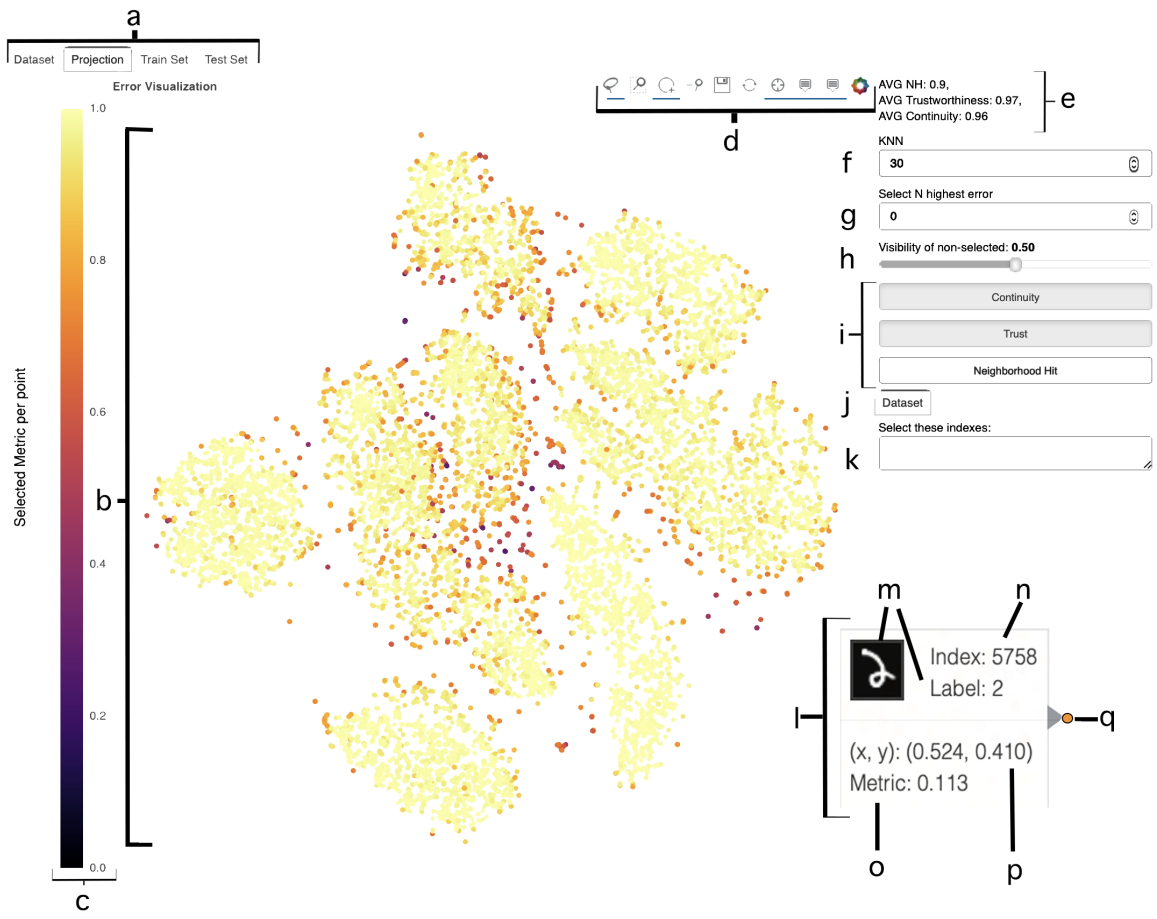


Figure 3.4: The dataset selection tabs (a), each showing a heat colormap of projected points (b) which are coloured according to color bar (c) as specified by the errors selected by error toggles (i). A range of interactions (d) is available to the user for this heat colormap. A user may click to select points (q) or move the mouse over a point to view the hover tooltip (l). The default hover conveys the class label (m), the index of the point (n), the x and y coordinates in 2d space (p) and the error metric defined by the toggles (o). Global metrics are shown (e) while the local metrics are computed at a scale of neighborhoods that contain (f) points. (g) and (k) allow selection by error or index respectively; non-selected points can be hidden by changing their transparency with (h). The Tools tab (k) has more usage in other views but remains consistently placed for similarity across views.

The user interactively selects the subset of the data they would like to investigate by selecting one of the view tabs Figure 3.4(a), either the 'Dataset', GT 'Projection', 'Training' set or 'Testing' set. The dataset view is the same as the start page in Figure 3.2 and this allows the user to recompute the model during the same session. The other views provide different perspectives of the training data for the user. In the training set view they can see the NNP projection. In the testing set view a user can see the generalizability of the model on the test set. In this way the user remains involved through each step of the exploration space.

Global Metrics and Global Tools

The Global metrics are shown in Figure 3.4(e) as a means of reducing the search space. If the user finds these satisfactory, they may move on to another sub-set of data. Alternatively if these are very low then something external has gone wrong in the training process and an adjustment of model parameters or checking the dataset may be more useful in reducing error than specific examination of the points with the tools listed here.

Finally Figure 3.4(j) allows the selection and usage of the specific subsets of tools, more are available in other views such as the training set view. Here we can see the 'Dataset' view, containing the selection text area. The components listed above are consistent across all views, but there are additional tools on other views. For example Training and Test set views have access to the tools which are contained within tabs such as "Distance" for finding neighbors or mapping by distance and "Visuals" for creating activation graphs. These tools will be demonstrated in future sections with examples of their usage.

Local Tools

The heat colormap is present in every view, but each is a unique projection. The heat colormap shows the projected points in Figure 3.4(b). This heat colormap is colored according to error per point, with darker points having higher error. The color scale 3.4(c) shows this gradient. This color range was chosen for two reasons. Firstly with the error points being the darkest, they form dark clusters in areas where the error is highest, giving intuition to the user at which severity these errors occur. A darker color for errors combined with a lighter color for low error makes the errors more pronounced and less obscured visually by the lighter colors. Secondly, choosing a lighter color for these lowest error points allows us to understate them in order to focus on our goal of fixing the error. The errors themselves are specified by user input of one or more toggle buttons in Figure 3.4(i). Each button when toggled 'on' enables a quality metric to contribute to the 'metric' score of the error per point. If multiple are selected they are aggregated by averaging, while a singular error metric toggled on is displayed as only that metric per point. Recall that the metric per point is computed based on a neighborhood around the point. Figure 3.4(f) is used to define how many neighbors are in the scope of this neighborhood and adjust the error accordingly. If no error metrics are 'toggled', the heat colormap displays discrete colours according to class label. For ease of use in isolating specific points Figure 3.4(g) and Figure 3.4(k) allow selection by error or index respectively. The non-selected points can be hidden as desired by changing their transparency with Figure 3.4(h).

Interacting with the heat colormap

A range of interactions is available to the user for the heat colormap in the toolbar shown in Figure 3.4(d); the first tool is a lasso selection tool. The user may select points using Figure 3.4(q) for isolation, at which point the selected point will become darker and all other points become less pronounced through the loss of opacity and outline. Selecting points is a key way to isolate points that they will perform further exploration on, making it a commonly used first step for other tools. The next button on toolbar Figure 3.4(d) is a box-zoom, which allows zooming in/out according to a user's interest scale, which prevents the large amount of points from cluttering the view. The distance between points scales by the zoom while the size of the points remains consistent with respect to screen size. In the case that points overlap, the zoom can be used to bypass the visual clutter to a more manageable scale where all points are visible. The third tool on toolbar Figure 3.4(d) is a tap tool, which is a more subtle selection method, selecting only

clicked points instead of by lasso. This is followed by a zoom out tool, to recover from any zooming in that the user has done. The fifth tool is a save, which exports the heat colormap and color bar to an image. Next we have the reset button, which undoes any interactions and reverts the graph back to its original state.
This gives the user the option to back-out at any time and fix unintended consequences. Finally, we have two speech bubble tooltips. These are the hover selections and the hovers can be toggled on or off depending if the user wants to see them or not.

Hover Tooltips

The default hover tooltip in Figure 3.4(l) conveys the class label in a visual and textual format to encourage trust in the model, as shown in Figure 3.4(m). The user can be assured that these are the points given in the training set and that training went as expected. The hover also contains the index of the particular training point in the dataset shown in Figure 3.4(n) which is useful for selection at a later point, as well as identification. The final information displayed on the hover are the x and y coordinates in 2d space as projected in Figure 3.4(p) and the error metric as defined by the user toggles in a numerical format for trust and transparency in the colormap.

The hover is intended to provide a qualitative understanding of the sample instances and their components, displaying the relationship between the class, input image and prediction. Each glyph requires its own hover to have differing information. For example the two displayed in the projection graph are for the class and metric heat colormaps. It makes no sense to have a metric showing on the hover Figure 3.4(l) when no metric is selected in class view, so it has its own hover-tooltip in Figure 3.4(d).

3.2 Identifying the Error

In this section we will answer the question: *'How can we understand the diffusion or limited quality in projection?'* by exploring how to determine a projection's quality. This is the subset of Figure 3.1 encapsulated under 'Find the error'. This model cannot be simply evaluated on a test set for accuracy like other deep learning models. Instead we use the metrics specified in Sec. 2.1.2 to outline in heat colormap form, the areas of error to see how error is spread over the 2d space. In this way we can qualitatively and quantitatively see where the errors are. We then focus on these points of error specifically while avoiding focus on the well-performing points to guide us towards an effective remedy to the diffusion.

The Heat Colormap

We will now examine the heat colormap that visualizes the hotspots of error and the context in which they occur. Knowing which points are worst affected by each error metric will allow us to know where we wish to focus our search for improvement.

3.2.1 What is the Error?

Recall the heat colormap displayed in Figure 3.4; this displays a local quality metric for each point color-coded as a score on a particular metric. This metric is selected by the user from the three metrics and computed at a local neighborhood. The neighborhood size is defined by a user input value in Figure 3.4(f). We call the inverse of a quality metric 'error'. The error refers to a low score of a particular metric, which then evaluates how badly a point is projected with regards to the neighborhood. This is in stark contrast with the kind of error in accuracy that is usually found in neural networks, as there is no easily 'correct' answer for where these points should go. If we were to evaluate the points based on their MAE for example, we would be left with the points that are furthest away from their t-SNE counterparts, but what we are actually interested in is not explicitly distance from t-SNE but rather whatever property causes points to become diffuse, whether far from t-SNE or not. We intend to use these hot spots of error as a means to focus our search. By using quality metrics, we open our search space to the possibilities of identifying errors across a variety of causes rather than just the failure to copy t-SNE exactly.

Class Labels

Without metrics, we show the heat colormap colored according to classes. This allows the user to make sense of the projection at a glance. Class labels are a more intuitive way to convey structure of the data, so we can see if NNP learned the structure by observing the class labels. Class labels are not provided to NNP for projection, training or inference; only for conveying the classes to the user. In Figure 3.5 we show the class view of MNIST in t-SNE.This serves as a means of showing the reader what they can expect from this dataset and intuit about the structure of each. In Figure 3.6 we do the same for the Dogs *vs* cats dataset.

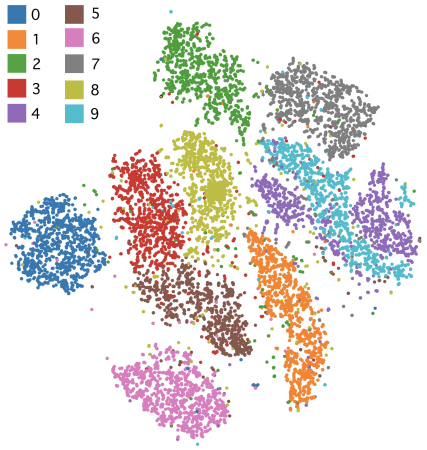


Figure 3.5: MNIST: Class View in t-SNE. Each color corresponds to a different class label.

Metric Comparison to t-SNE

In this section, we briefly compare the metrics on t-SNE to NNP to verify that the error is consistent across both methods of projection.
For all three cases listed in Figure 3.7, t-SNE and NNP share the same error placements, but with NNP having comparatively more. NNP is good at mimicking t-SNE's success, but not the scale of its flaws. This is expected due to the differences in the global quality metrics but is important to note as we can draw two conclusions from this. Firstly, that NNP captures the structure of t-SNE. We can see this from the class labels, as well as the layout of the hot-spots of error.

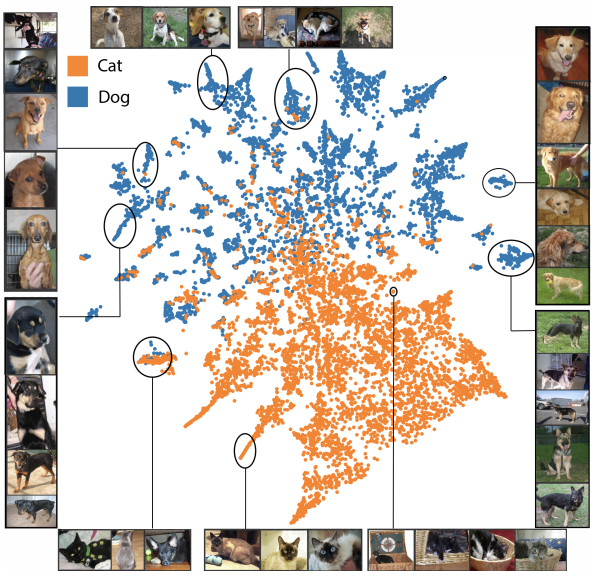


Figure 3.6: Dogs *vs* Cats: Class View in t-SNE.

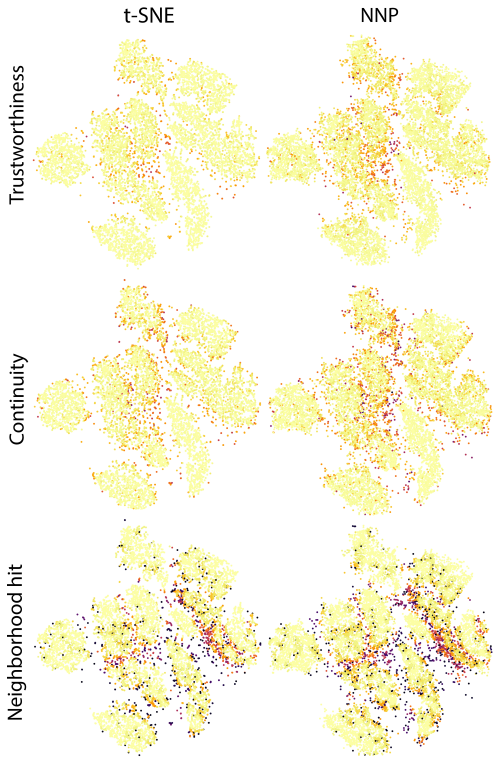


Figure 3.7: Error hot spots in t-SNE and NNP. Darker/more red areas show lower metrics. Notice the similar placements of the darker points, indicating training success.

This is not evident, as projection styles often have very different errors to those outlined here [21] and the structure of clusters could be formed in the same general position without having the same kinds of errors at a fine-grain scale. This suggests that NNP has these errors in the same places as t-SNE because NNP is capturing some mechanism of the ground truth projection, rather than trying to capture the effect of t-SNE in a completely different way. Secondly and more crucial for our goal, the error is of a similar scope to t-SNE. We have error in the same areas so if we reduce the error we are likely to see a much closer similarity and therefore competition, which is our main objective.

Continuity C

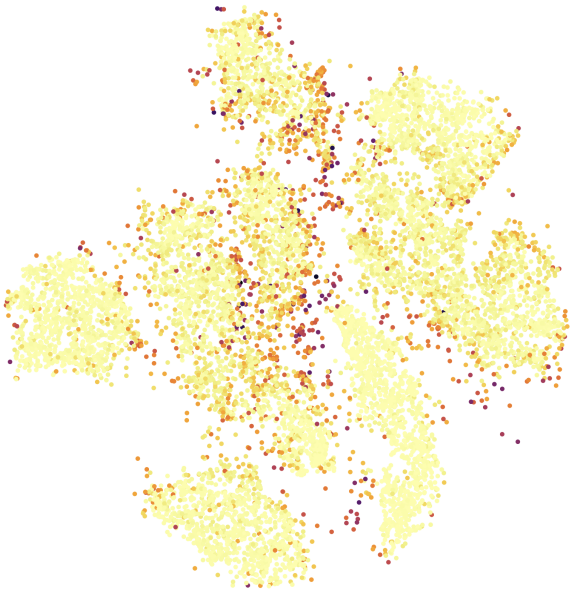


Figure 3.8: Continuity heat colormap

Continuity (formally defined in Sec. 2.1.2) measures how many points in the projection neighborhood of a point should actually be in that neighborhood, or are falsely shown as neighbors. As visible in Figure 3.8, these points tend to appear between or on the edges of clusters. If being closer to the center of the cluster signifies a solid identity within that cluster, then these points are the ones who are not sure of their identity and are among other points who are not sure, leading to neighbors in the projection that should not be. The lowest score of continuity corresponds with the points in the midpoints between clusters and on the border, this is the same position that we see the diffusion in points.

Trustworthiness T

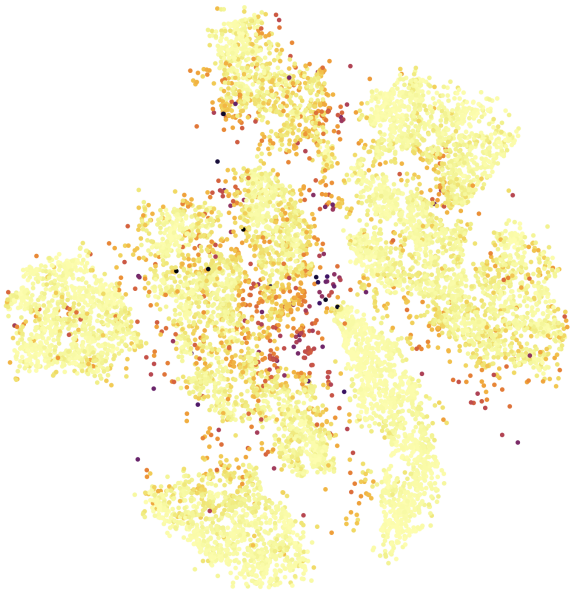


Figure 3.9: Trustworthiness heat colormap. Darker/more red areas show lower metrics.

Trustworthiness (formally defined in Sec. 2.1.2) measures how many neighbors in the data are missing from the projection neighborhood. We see a similar distribution of error in 3.9 to those in the previous Figure 3.8; the worst scoring points are those between clusters and on the borders. Trustworthiness seems to more harshly penalize certain points in the projection, particularly those between multiple clusters like in the top center of the heat colormap, while less harshly rating those that are within clusters, evidenced by the black stripe of dots in the left of the center cluster present in continuity but absent in trustworthiness. With these two metrics examined, we can see the case for aggregation. Both metrics target the diffusion harshly, while calming the areas only present in one, creating a smoothing factor.

Neighborhood Hit NH

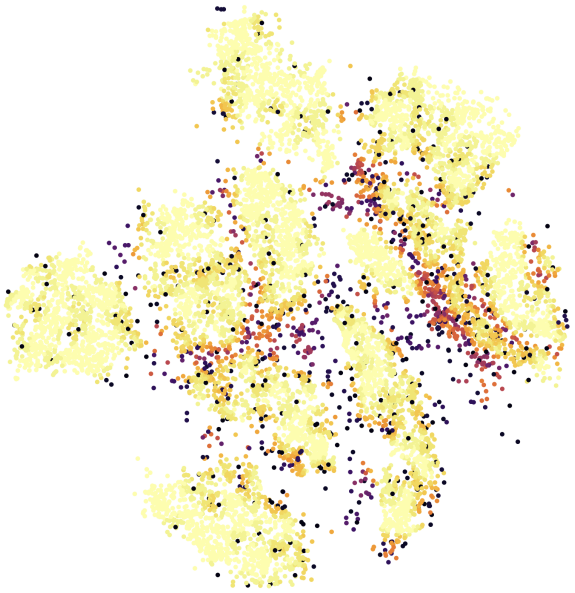


Figure 3.10: Neighborhood hit heat colormap. Darker/more red areas show lower metrics.

Neighborhood hit (formally defined in Sec. 2.1.2) is a metric that is very harsh on the points between clusters. This is expected as cluster separation is exactly what neighborhood hit measures. The severity of this harshness is shown in 3.10, where much darker points can be seen than in the Figures 3.8-3.9. Neighborhood hit may sound like it is the perfect metric for judging similarity to t-SNE which prioritises neighborhood preservation, however this metric also strongly negatively emphasises 'misclassifications' that appear in both t-SNE and NNP.
That is, points which are similar to their cluster in the input, but share a different class label to those points, *e.g.*: A 1 that is indistinguishable from a 7.

Why not be harsh on misclassifications?
Since we are ideally trying to match t-SNE, we ignore cases that are incorrect in both NNP and in t-SNE. Focusing instead on areas to improve to meet the level of t-SNE. We may have such cases where both projections are correct in placing this point as a misclassification, which is most likely the behavior we see in Figure 3.10. A point may be correct with respect to D in $P(D)$ but because the class labels, which the projection is unaware of, are different the neighborhood hit decreases significantly. This means that sometimes neighborhood hit can be detrimental to the kind of behavior we want to see.

Aggregate Metrics

So far we have identified metrics that are able to quantify the diffusion that we want to remove, however some of these are too harsh on non-diffuse parts of the projection. This harshness appears as distracting dark spots that we do not want to focus on when isolating diffusion. By aggregating two or more metrics, we are able to compound the harsh penalization of diffuse points, while points which are not diffuse are weighed on their performance between metrics. The higher score in one metric of points that are low in others, will bring the score of those points to a higher total value and allow the discrimination of the diffuse points which are low in all metrics.

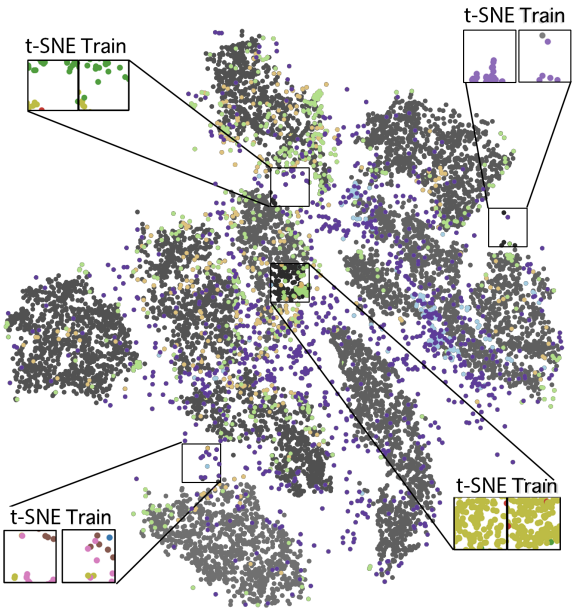


Figure 3.11: Exclusivity of quality metrics: continuity (Green), trustworthiness (Orange), neighborhood hit (Blue) and aggregate of 3 (Purple)

In Figure 3.11 we demonstrate the usage of the aggregated metrics compared to each metric when

computed by itself, for this diagram we chose to show the training projection in gray-scale to easier emphasize which points are selected by which metric exclusively, *i.e.* by only one and no others. We take the 10% lowest scores for each metric, and subtract from these the points which are in the lowest 10% of other metrics, including aggregate metrics. Green are the lowest 10% of continuity that are only selected by continuity and not by other metrics. Orange are the lowest 10% of points highlighted only by trustworthiness, blue are the lowest 10% of neighborhood hit and purple are the lowest 10% only selected by an aggregate of all three metrics.
We highlight four situations in this graph, which fit into two cases. The center most inset illustrates the case of points that are 'smoothed' by the aggregation. These are points that are picked up by some error metric that are not desired when focusing on diffusion. In this case we can compare the training set and t-SNE and find no obvious diffusion.

In the other case, we see points that are not considered the lowest 10% of metrics until all three are aggregated. These cases demonstrate where diffuse points exist by showing the insets of direct comparison between t-SNE and the training set. We see many scattered points that are split from clusters, that only are selected by the purple aggregate. In this way we more concretely isolate the diffusion, as illustrated by the purple dots in the figure mostly lying in the diffuse areas between clusters.

Neighborhood size for Metrics

The neighborhood size we use in this thesis is the same value as the t-SNE perplexity. This is simply a rule of thumb, and could be set to any integer value with the caveat that a neighborhood too small means that small groups of badly placed points escape detection by clustering and forming entire neighborhoods. A size too large means that inter-cluster dependencies form in the metric and we start evaluating the metrics of points that belong in a small cluster by the nearest points outside of the cluster as well, regardless of distance or similarity.

3.2.2 The Error is Diffusion

In this chapter we showed that we have found that the errors occur at the same places as the diffusion that we wish to reduce. Some metrics are better than others when alone, but aggregation compounds the effectiveness of the error of interest to more concretely outline this diffusion. We found that the error lies in the same place as t-SNE's error, which is important for establishing that improving these metrics will bring us closer to t-SNE. With these two pieces of information, we know that we have found the error that we want to reduce and that if we do so we can expect to behave more like the t-SNE ground truth. So now we can focus on this error and try to reduce it to reach our goal of making NNP more competitive with t-SNE.
We can answer the sub-question: *On which points does NNP work well?* Simply, these are the same places that t-SNE performs well, and provides a good example to learn from.

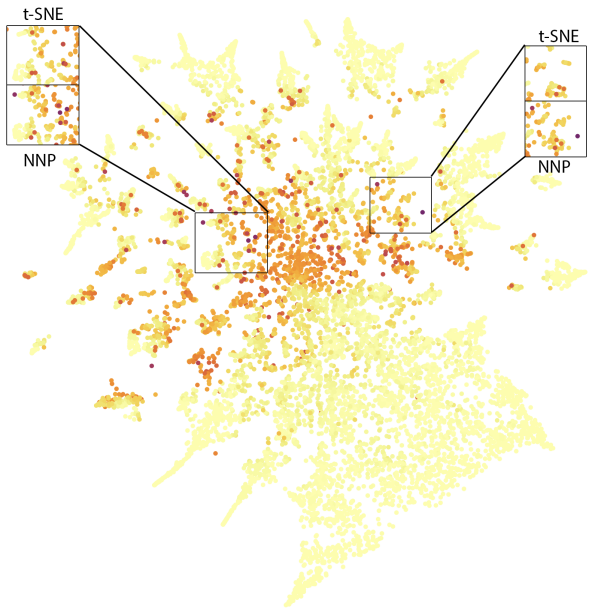


Figure 3.12: Diffusion within Dogs *vs* Cats. Insets show diffusion compared to t-SNE. Darker/more red areas show lower metrics.

3.3 Isolating the Error

With the error heat colormaps defined in Section 3.2.1 we now know where the largest errors are implicitly. The threshold tool aims to explicitly isolate the error, so that we can focus our attention into these areas. This focus narrows down the space of causes to those that only occur in the error that we want to reduce to improve the projections.

Although not traditionally applied to a dataset before establishing a baseline as in Figure 3.1, we will use the isolation tools to explore the points as a first exposure to the problem. Usually we rely on only the class labels and quality metrics to determine our baseline. We will apply these additional tools to the training set in order to demonstrate the usage of locating, selecting and isolating error and to become more familiar with the data.

3.3.1 The Thresholding Tool

The process of selecting the lowest performing point is trivial; the user simply inputs a number in Figure 3.4(g). The threshold operation then takes the current selected metric that is applied to the heat colormap, finds and then selects the smallest scores in this metric. This selection invokes the same behavior as that of the lasso tool in Figure 3.4(d). In this way all selected points are treated as equals, whether user or threshold selected. This is useful so that the user is able to manually select through brushing instead of by threshold if they wish to use a different subset of points for further steps, giving them much more freedom to explore.

Selection and Isolation

We can use a threshold parameter to reduce the error we see to only the highest percentile.
For the purposes of this experiment, to demonstrate the tools usage, we find the worst performing point in the test set. This performance is according to the neighborhood hit, and a metric neighborhood size of 30. If we can isolate and focus on this point, we aim to find a reason for it's low metrics. Given the layout of error in Figure 3.11 we can say that the error metric is related to the diffusion in points. We aim to reduce diffusion by finding the same problems that reduce the quality metrics and increase error.

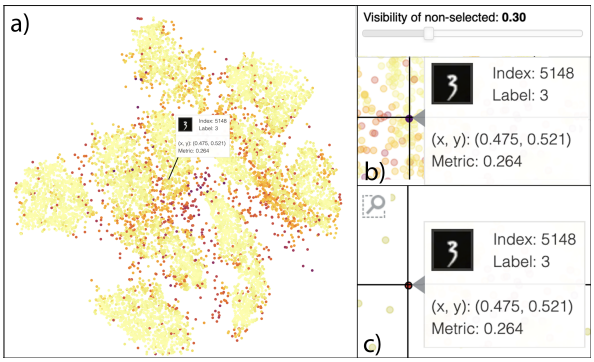


Figure 3.13: Two main tools for focus used on the same point (a), visibility slider (b) and zoom tool (c)

This particular point is a big contributor to error by being the largest error in the projection.

Once selected the points can be isolated in two main ways, shown in Figure 3.13 these ways can be used together if desired but mostly depend on context.

Isolation by Opacity

If the user wishes to keep the global context of the point in mind, the *visibility slider* Figure 3.13(b) will control the opacity of all non-selected points. The user can make all uninteresting points increasingly transparent until they can focus on the selected points. The non-selected points can be tuned to a desired transparency so that they still provide a context to the selected points, as shown in Figure 3.14.

Isolation by Zoom

If the user wants to examine the neighborhood context by fully focusing on a single neighborhood they can use the *zoom tool* in Figure 3.13(c), allowing a more precise movement between points and usage of tools without the other points cluttering the view. Any points will scale their metric color according to view, allowing the user to easily see how points in a neighborhood relate to each other.This color scaling allows the user to focus on a specific scope of the projection. The user is able to move around at this local scale through the help of a panning tool , and can also easily reset the projection to the original context through the reset button.

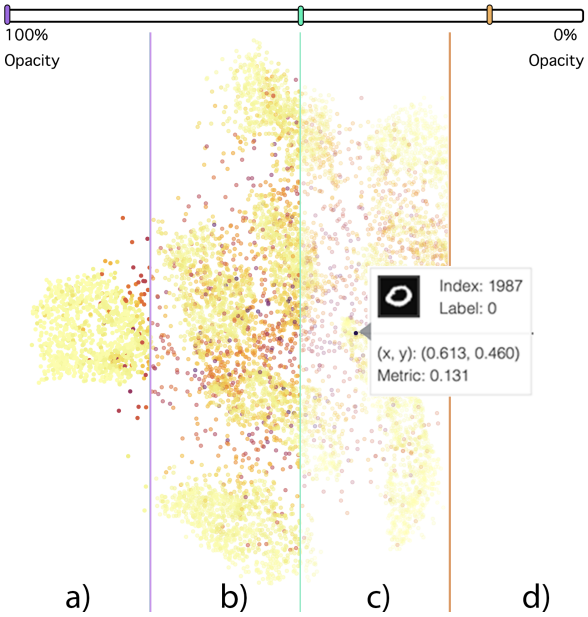


Figure 3.14: Slices of the visibility slider used to set the opacity of the points to 75%(a), 50%(b), 25% (c), 0%(d).

Both the reset button and panning tool are found in the toolbar (Fig. 3.4(d)).

Usage of the Zoom Tool:
The usage of the zoom tool is demonstrated in Figure 3.15. In this figure we see the context of a point in the projection (a), we then zoom into this point's neighborhood (b) and are able to see the relative colors of its neighbors. Finally we use the zoom tool again to see the point by itself in (c)

3.3.2 Examining the Network's Decisions: Method and Baseline

From the previous subsections, we learned how to use multiple avenues of selection and isolation. We apply these and find that the lowest performing point in neighborhood hit is a 'misclassified' 0. This point is 'misclassified' because it is placed within a cluster of 1s, leading to it's low quality metrics. The nearest neighbors of this point by proximity in the projection do not seem to share any easily identifiable features with it, as far as we can see in Figure 3.16 and it is not an obviously misleading case of a 0 that looks like a 1. There is no similar 'misclassification' in the t-SNE projection, as is the case with those errors that are present in both t-SNE and NNP projections that we justify ignoring in Sec. 3.2.1. In fact no single class 0 point is placed into this cluster in the original t-SNE, although because this is a *test sample* we cannot say for sure where exactly this point would or would not be placed by t-SNE as we are unable to stably generate a test projection.

We want to answer the sub-question: *On which points does NNP work poorly?* At this point we know that NNP performs poorly on the point selected in Sec. 3.3.1, but we do not know which properties of this point places it within that category. We want to know more about this point, but simply looking at the neighborhood, metrics and hover is not enough. What we have done so far is look at the results of the projection and compare them to what we have seen before. What we want to do is look inside the black box rather than reason about what comes out of it. We want to be able to tell the diffuse points apart from the well performing points by finding some discriminatory property within the input or network behavior.

These two experiments help build a foundation for opening the black box. We have justified our metrics as being helpful in identifying error, which is in turn helpful for seeing if we improved the projection. Furthermore, the hover, threshold and selection tools that we have established are widely used in our experiments and will be useful when investigating these points in the next sections.

We were able to isolate a point as outlined by the error metrics in the previous section and we were able to focus and dive down into this point to investigate it. We found some clues and made some conclusions based on this, but have no actionable information to improve the projection from the usage of the tools we have so far. We must use more complex tools to get into the decision process of the neural network.

At this stage, we have examined some of the largest errors superficially, but we want to go more in depth. Now we introduce some tools to focus explicitly on investigating NNP's decision process. We do this to gain intuition from the similar samples in the data.

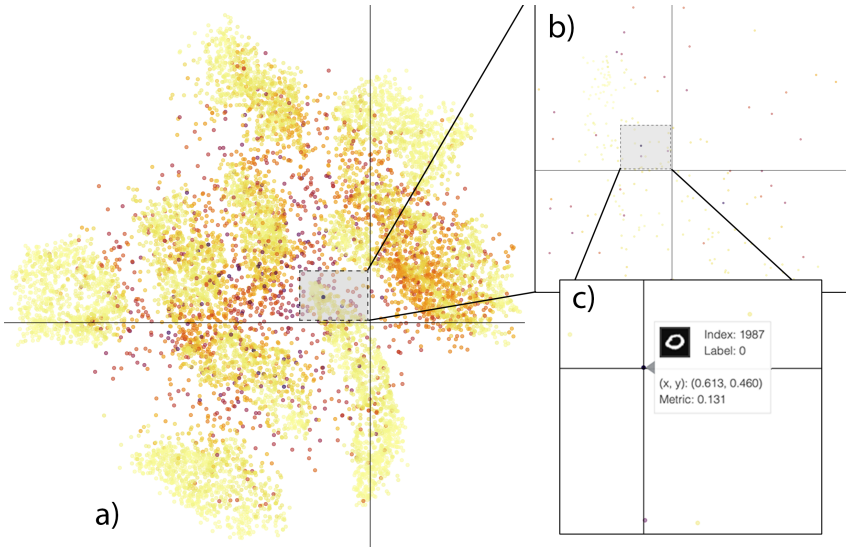


Figure 3.15: The zoom tool to focus on the point (a), inlets show each subsequent magnification. (b = x1, c = x2)

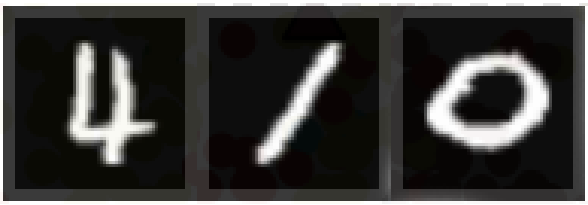


Figure 3.16: The pixel values of the worst point (right) and its two closest neighbors outlined so far accessed by the hover tool

In this experiment we will determine why a specific point was placed in a particular location by tracing the choices back to training and the characteristics of the input provided. We focus on the input and its relation to the diffusion in order to narrow down if something in the input directly influences the diffusion, as opposed to something that is part of the network architecture.

For this purpose we will be using two main tools: the nearest neighbors tool, to see which points are most similar according to the network, and the interpolation tool which allows us to see how far a point has moved from its placement in the ground truth t-SNE projection.
The nearest neighbors tool allows us to determine if the network judges a point to be similar to another and placed nearby, where we would expect similar points to be, therefore demonstrating if the learned features are correct . The interpolation tool allows us to see if the training struggled to grasp some aspect of the original projection by showing the disagreement between the predicted NNP and actual t-SNE values, therefore demonstrating that the learning did not succeed completely.

Before we apply these tools, we explain the logic and formulation of them:

3.3.3 Nearest Neighbors Tool

The nearest neighbors tool provides a method of answering: *What did the network learn?* by demonstrating the similarity of points across datasets. The tool takes a number from the user, as well as a subset of the points selected by any selection method. The selected test point activations are compared for similarity with the other training point's activations, and the most similar activations are returned. In this way we let NNP tell us which of the training-set points had the strongest influence in determining where a test-point was placed by interpreting its learned representations [14].

For our experiments we will be using the nearest neighbor tool to select the 30 nearest neighbors.

This will allow us to get some idea of the neighborhoods in D that are being evaluated for the scores we are seeing.

For a user, a nearest neighbors tool in the training set indicates points that we should like to see together if the network has learned them correctly. A nearest neighbors tool used in the test set indicates points that show what the network has learned. In both circumstances we would like the points that are close together to be similar, to indicate that the network has learned the correct properties. However, since the network learns weights globally, it can be expected that not every point fits perfectly as a result of the weights fitting the majority rather than a specialized case.

Intended Usage is Iterative

It is important to note that the neighbors are added in order of similarity during normal usage. In most experiments within this thesis we do not show the gradual addition of neighbors. The most informative use is to use the incremental counter on the neighbors tool to add one at a time, thereby getting a feel for the positions of neighbors in similarity.

3.3.4 Extracting Learned Representations vs Euclidean Distance

The nearest neighbors tool can compare the train and test set at the same time through differing inputs to specify from which set to fetch how many neighbors from. A user may select a test sample, find the nearest neighbor to this test sample, the nearest training sample and visa versa.

The Relevant Neighbors

For comparisons between the training and test set we care the most about how the network interprets their similarity. For the other views looking at their same subset of data, *i.e.* the training set view looking for closest training samples, the nearest neighbors tool does not use the neuron activations for comparison. In these cases we are showing the closest samples in that subset of data by Euclidean distance. This is because the most similar activation within one subset of the data, is already shown implicitly through the train or test graph as nearby predictions.

How to Extract Activations

Retrieving the activations for comparison is not obvious, since the activations are part of the black box mechanism that is obscured. To do this we create a surrogate model of the NNP model which was used to create the projection. We give this model a duplication of the activations and weights so that the model is functionally the same except that we remove the output layer. This leaves the sub-final layer exposed to output the activation of the layer before the final processing. We can then use this headless model to predict the activations, as the output is now the untouched computation of the layer without final activation. Once we have computed these activations, we use sorted Euclidean distance between activations to compute the most similar activation and this is treated as the most similar sample according to the network's decision process which it has learned from the input.

Nearest Neighbors Visualization

These nearest neighbors in D are placed over their respective coordinates in $P(D)$ so that the user can see where the projection has placed them. To differentiate these points they are marked with a triangle glyph instead of the usual circular glyph, but remain colored according to the metric the user has selected.

Nearest neighbors from the test set, when used in both views are coloured by their class label. Firstly this allows them to be differentiated from the nearest training neighbors when viewed on the same projection. Secondly, although we do not rely on or prioritize class labels, they serve as a way for the user to tell at a glance how the network learned a sample. If the nearest test samples vary in color and placement, we know that we have failed to learn some property that defines that sample.

A Caution of Nearby Test Samples

The third and final reason we do not show the metrics of the nearest neighbors in the test set is that we know that the test set is a generalization of the training set. Problems in the training set will propagate to the test set; moreover we can only hope to focus on problems within the training set. If we see a point that we want to investigate, we can examine it's nearest training set partners and see if they

score badly to determine a problem; we can hypothesise about what was or wasn't learned about the sample, what the network thinks is similar, what t-SNE thinks is similar and what is similar. If we instead try to examine the most similar test points and see that they also score badly, we are unable to draw conclusions from this about our network other than this point and ones that are similar to it score badly. We cannot compare the test to the ground truth and if we want to try understand the decision behind this test point we must examine the training set anyway. Thus, the test neighbors largely function as a way to examine if the point is confusing or unfamiliar rather than a more informative exploration.

3.3.5 Interpolation Tool

The interpolation tool shows the distance between the ground truth t-SNE projection, and that same sample's placement in the NNP training projection. This distance provides insight into answering *What did the network not learn?* by means of a comparison between t-SNE and NNP samples and specifically identify the disagreements between these two training sets.

Interpolation Formula

The formula for the interpolation is:

$$X_i(t) = (1-t)x_i^{nn} + tx_i^{gt} \tag{3.1}$$

where $X_i(t)$ is the new placement of the i-th training point in x or y coordinate. x_i^{nn} is the position of the i-th training point in NNP. x_i^{gt} is the position of the i-th training point in the t-SNE projection, while $0 \leq t \leq 1$ is the current value of the slider as placed by the user.

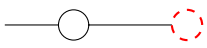


Figure 3.17: A training sample on a trail, $t/1$ of the way to that same point's placement in GT t-SNE (red). Note the t-SNE point is not visible.

Interpolation Tool 1: Slider

The interpolation tool can be used in two ways. The first usage is a slider where $t = 0$ corresponds to the placement in the NNP projection and $t = 1$ is the placement in the t-SNE projection. The user can move the slider in real-time to create an animation between the location of the points to understand their movement. This is useful for interpretability, as the user can move the points in real time and judge the distance by the relative speed and difference covered by a point at each step of the slider.

Interpolation Tool 2: Trails

The other usage is a toggleable path, which simply draws a line between the two points at $t = 0$ and $t = 1$ of the slider. This path is much more useful when tracking multiple points, as it can be hard to keep track of many moving points, especially when they may overlap or cross each other. It is also the only one that conveys itself well in images, and as such will be the main usage within this thesis. Figure 3.17 demonstrates the the notation of the visualisations used by the tool.

Interpolation Trail Visualization

The path is colored by the magnitude of the difference of distance in the two points. Dark blue means that the NNP point is much lower than its t-SNE counterpart in x and y. Dark red means NNP places this point much higher in x and y. If a point is about the same placement the line is white, but will likely not be visible as the length will be too small. The trails do not use the metric of the points as a color for two reasons; first of all the metric score of a point in t-SNE and NNP may differ and more importantly, the light yellow which serves to draw the user's attention away from the well performing points, does not stand out when used for the trails and the tool becomes very difficult to use.

For a user, the trails indicate a discrepancy between NNP and t-SNE. Long trails indicate that NNP chose to place this point far away from t-SNE, as a product of the learning. Either the function computed by NNP was not specific enough, restricting placement based on the computation of other points or NNP learned something different. In the case that trails are long and cross over each other NNP may be trying to preserve the distances between these similar points while t-SNE does not.

Interpolation with the Nearest Neighbors Tool

In the test set the interpolation tool behaves slightly differently as no direct comparison to t-SNE can be made. Instead the user may use the Nearest neighbors tool explained above to show points from the training set, which can then be used with the interpolation tool above in the same way. Yet again to differentiate these points they will be marked with a triangle glyph, while the points belonging to that view remain circles. This can be seen in Figure 3.18 where a training point is placed as a triangle near its nearest test set in the test view and can then be compared to t-SNE.

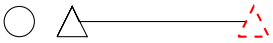


Figure 3.18: A test sample (circle) and its nearest training sample (triangle) placed by the nearest neighbors tool in the test set projection, followed by the trail leading to that same training point's placement in GT t-SNE (red)

3.4 Examining the Learning: Ensuring Correct Training

3.4.1 Establish a Baseline

We have introduced the basic tools that we use in our workflow (Figure 3.1). We are now able to apply these to go deeper into the points isolated in the same way as in the previous section. Before we go deeper, we should ensure that the biggest problem we have is actually deeply hidden, not a superficial mistake caused by an implementation error. To ensure this, as the first step to evaluating any projection, is to check if the training went well. We expect that the visualization of cluster separation is better in the training set because we have explicit examples to learn from. If the training is not successful there is little hope that the model will be able to generalize well in the test set, so we set a baseline by examining the training set for obvious problems. Since we are examining the training set first, we do not need to examine activations only Euclidean distances, as the most similar activations are conveyed through the nearest neighbors in the projection and there is no need to examine test samples.

We will theorize as to what makes these points behave how they do. We do this to gain intuition as to the kinds of situations we are likely to find. Since we will examine what NNP has learned in future sections, we will focus here on what is being taught; how the training process went, and what we think t-SNE has seen versus what it appears NNP has seen based on each decision process.

Step 1: Select and Isolate

We first start by applying the threshold tool to specify the worst $n = 1$ training points, according to the aggregate metrics. We do this by making sure that all of the metrics are toggled on and putting a value of 1 into the threshold input.

The tool returns a point, but it may be difficult to see without searching. The point is selected as part of the return of the threshold tool, so to help us to find the point we can use the visibility slider (Introduced in Section 3.3.1).

We set the undesired points to a level of transparency where we can clearly focus on our selected point and then use the hover tool to get some idea of this point.

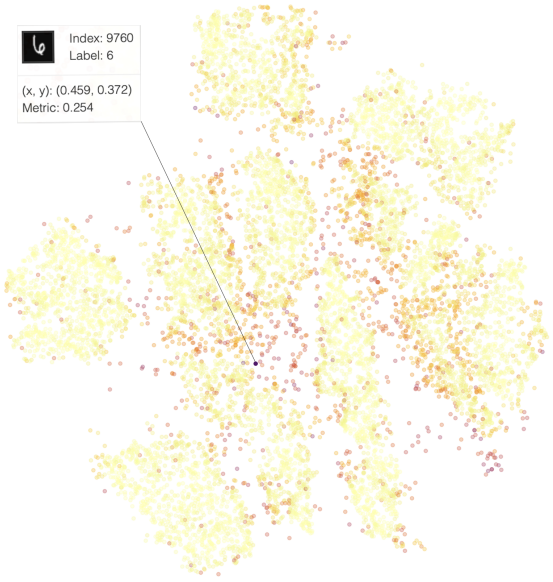


Figure 3.19: Step 1: Using the selection tool and visibility slider to find the worst training point

Using the hover tool in Figure 3.19, we know that this point is index 9760, with the label 6. The image attached to the tool confirms that this is a 6 and does not seem to be a difficult case. This point is between the clusters, so it is part of the diffusion.

Step 2: Find the Nearest Neighbors

We then move over to the distance tab in training which is placed at the same position as Figure 3.4(j). We simply input the amount of neighbors $k = 30$ into the input specifying 'nearest train neighbors'.

Each of the most similar neighbors in D as evaluated through Euclidean distance appear on the heat colormap as a triangle glyph in their positions as projected by $Pnn(D_s)$.

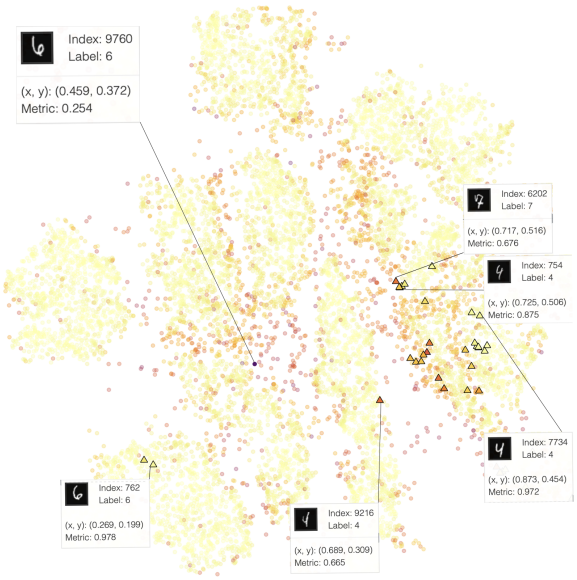


Figure 3.20: Step 2: Using the nearest neighbors tool to examine the most similar points in Euclidean distance

We are able to evaluate this point objectively with regards to its metric, as we can compare the metric in the hover tool to that of its neighbors. This worst point is very low scoring compared to it's 30 nearest neighbors, and of those neighbors the lowest scoring points seem to be much more difficult cases than our worst point 9760. The first thing to note here, is that these points seem to be from varying classes, so it may be that this point is dissimilar from other points in its class and therefore pulled away towards other classes. Of its 30 nearest neighbors only 2 are sixes in the cluster we would expect. Secondly the placement of these neighbors is very spread out, this point appears to be placed in the center of these points as a result of placement at a midpoint between groups of neighbors.

Evidence of a First Cause

We have found that this point appears to be at a midpoint of its furthest neighbors. We would assume it would like to preserve the distance globally between points, but un-intuitively it does not scale with proximity, *i.e.*, the closest neighbors do not appear to have more weight than the further neighbors when determining distance, nor does quantity of neighbors. If proximity were the case we would expect to see the point closer to the sixes, since they are the closest neighbors (seen through iteratively adding neighbors). If quantity was able to weigh the point closer, in Figure 3.20 we would expect to see the point placed 28/30 of the way to the cluster on the right as 28 of its neighbors are there. In absence of both of these, we hypothesize that this point is pushed further away from its nearby samples by a form of global distance preservation but one that does not focus explicitly on placing at a midpoint, but rather maintaining global distance between two groups of very dissimilar points while placing a point that is equally far from both. However we do see this midpoint case quite frequently, so we will have to explicitly disprove it before it is removed from our potential causes.

Step 3: Interpolate to Ground Truth

For this section we will be relying on the trails of the interpolation tool. The usage of these trails begins by toggling them on, which is as simple as clicking a button below the nearest neighbors tool. Once this is done trails will be visible from the nearest neighbor triangle glyph to the place where that point is placed in the ground truth projection (t-SNE). In Figure 3.21 we can see that this point in t-SNE is much closer to the other neighbors than to the ones that arguably, should be more similar given the class label.

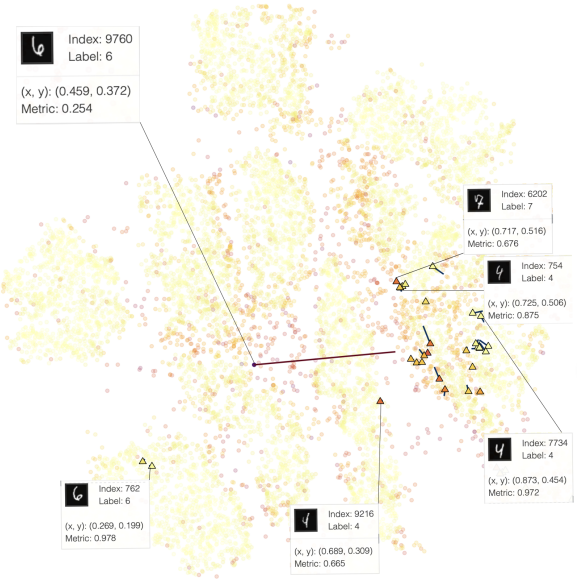


Figure 3.21: Step 3: Using the interpolation tool to examine the similarity in training (NNP) and ground truth (t-SNE)

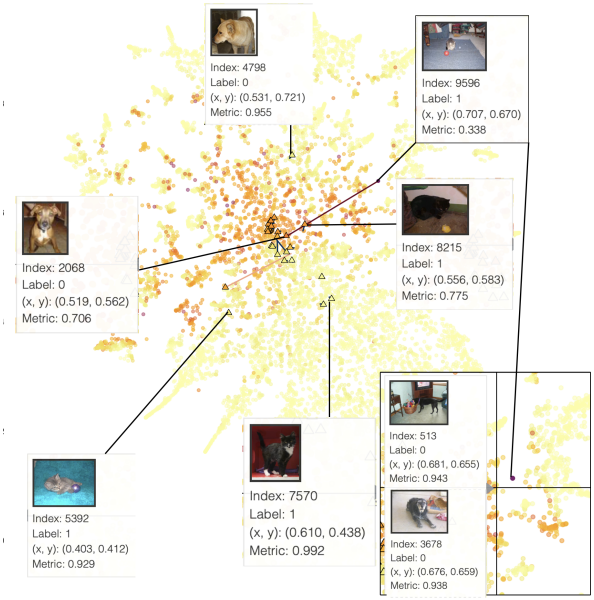


Figure 3.22: All three steps mentioned previously, applied to Dogs *vs* Cats training projection. Inset shows instances in the nearest cluster to NNP placement.

t-SNE is paying attention to the local scale of neighborhoods when projecting a point and the similarity computation seems to have picked up that the 6 appears similar to an upside-down version of the points in the right cluster. Since NNP learns abstractions for all of the points, it may be learning similar features instead, a kind of 'definition' of what makes a six across all sixes, that t-SNE does not see by similarity computation. This is evidenced by NNP trying to place the features it sees in point 9760 near points that it sees as similar in the 6 cluster, pulling it in that direction. This seems to point to a problem in the architecture of NNP that restricts the function of the network, forcing it to place this point further apart from similar samples.

Application on Dogs *vs* Cats

Now we apply the same steps in the previous section to another dataset, to demonstrate the usage of other datasets and to give some insight into the network's decisions in a different scenario.

In Figure 3.22, we see the worst performing point in the Dogs *vs* Cats dataset, is a cat that NNP places near a cluster of dogs, while t-SNE places it right in the center where the two classes merge. This t-SNE placement may not teach NNP a clear class identity for this point. We notice that NNP is placing this sample near other points where the animal is fairly small in the picture while t-SNE just seems to have placed all of the 'strange' examples together in the most error part of the projection (visible as the hot-spot in error in Figure 3.12). Examples of strange points within the center cluster include a clip art image of a cat with a mailbox, and a image of a dog framed in a star shape (Fig. 3.23(c). The images that t-SNE sees as most similar are very small in resolution, while NNP may have focused on objects in the photos or the relative size of the animal in frame (Fig. 3.23(b). There may be some artefact that low resolution images have from the feature extraction model that t-SNE is seeing as a feature. We receive the input data from the VGG16 model, which extracts the features using a model that is trained on a very large dataset to recognize images. We do not know the features that the model has to work with; it would seem NNP is putting more weight on features with objects such as toys.

The feature extraction model is trained on images of all kind, not explicitly of pets so it may identify objects easier.
The point's placement in t-SNE and Euclidean distance nearest neighbor point 4798 seem to have nothing in common. This suggests that this point is a difficult case, which brings uncertainty to the evaluation of training performance. Recall that we can judge how well training went with our global quality metrics. When examining these quality metrics we find they are very high and NNP matches t-SNE excepting a -0.01 to trustworthiness. Clusters and error distribution are the same across NNP and t-SNE so we conclude that the training was sufficient. Again we see that there seems to be a mismatch in how the two projections make decisions; t-SNE wanted to keep the point close to its most similar points so desperately that it formed a cluster of strange points in the middle, while NNP seems to have learned something else completely.

3.4.2 Successful Training

We performed these experiments in this chapter to set a baseline and make sure that training went as planned. There is no evidence of a false signal here causing undesirable correlations. In MNIST, the decisions made by the network seem to make sense and are quite easy to interpret. The clusters seem to have formed well as for the most part classes are grouped together. Our worst performing point appears to be a case of mismatch between the local and global methods of t-SNE and NNP rather than a mistake in training. This is not an actionable point as of yet, but rather a point of interest to remember for later. For Dogs *vs* Cats we see the same situation as in MNIST, except this time we are unable to justify this as a midpoint between what was considered similar, which is evidence against the first part of the theory in Sec. 3.4.1. Another potential reason for this could be that the input is too far from what the network has learned, making NNP place it randomly or with emphasis on very distant similarities.
In the next chapter we apply these tools again and some new ones in a more advanced experiment to further investigate the results found here. With this baseline established here we know that we have a working implementation of NNP, which did learn and is performing as expected. So we can finally answer: *On which points does NNP work poorly?* by examining the points in the next chapter, where they are without the guidance of t-SNE and must rely on learned features.

(a) The worst performing point

(b) Most similar point in NNP (left) and t-SNE (right)

(c) Two samples within the center cluster that t-SNE considers similar to point (a)

Figure 3.23: NNP considering objects in photos while t-SNE seemingly considers resolution

3.5 Examining learned features: Influence of Input on Generalization

In the previous section we concluded that training went suitably; we did this by examining the the global metrics in comparison to t-SNE and examining structure in class labels.
We also went into some detail on the worst performing points and made some assumptions about the decisions that NNP made to place these specific samples. Such assumptions are not widely applicable, because we focused on the identity of those specific points. We did see some indicators of potential causes of error that could be persistent across all subsets of data.
Some of these causes that may be more significant are:

- A mismatch between t-SNE and NNP that manifests as either a repulsion of points in NNP, or the placement of a point at a midpoint of neighbors.

- Distance from the known or unfamiliarity with samples confusing NNP.

In this chapter we try to isolate and solidify these potential causes, moving from the training view to the testing view in our workflow (Figure 3.1).

Why Examine the Test Set and the Input?

We examine the test view to get an idea of inference as the test set more clearly shows the decision process without guidance. In the training view, the projection is told to place a point based on a given label of coordinates; with the test set it can only apply learned knowledge. We examine the input because we break our network into two components when trying to examine potential problems, the first is the input and the second is the network architecture.

We examine the performance of inference with regard to input so that we can isolate the causes that occur because of input so that we can deal with them separately.

How do we Examine the Input through Inference?

Each test point (selected here as the worst performing point) is placed based on the features learned during training. Therefore, every inference decision can be traced back to an observation seen during learning. In other words the input data told the network to infer this way in the test set. We break down the placement of each point to a combination of its training samples through the nearest neighbor tool, as training samples are the input processed through the network.

Experimental Setup

Now we will apply the tools in the previous chapter, along with the nearest neighbors tool for test samples; recall that before we only used it to see training neighbors. We apply the same line of queries to the test set: locate error, select points, isolate, examine and find neighbors. We do this in order to examine the effect of input on the generalization of the network and discover what properties of input control the inference and placement of an unseen test sample. We will attempt to use this new understanding of placement to answer: *On which points does NNP work poorly?*

Since we explained the functionality of the nearest neighbor and thresholding tool in the previous section, all figures from this point forward will contain steps 1-3 from the previous section in one figure, to show the selection of the error point, its neighbors and their comparison to ground truth. We will then use an additional figure to show the comparison with the test set neighbors which we haven't explored before.

3.5.1 Generalization in MNIST

The Worst Point in MNIST

In Figure 3.24 we can see point 7945 which is the worst performing in the test set of MNIST, according to an aggregate of the metrics. This point lies at the midpoint of its 30 nearest neighbors by activation distance; that means that NNP considers these 30 points the most similar according to what it has learned. This midpoint aligns with a hypothesis we had in Sec. 3.4.1, where there seems to be repulsion from similar points.

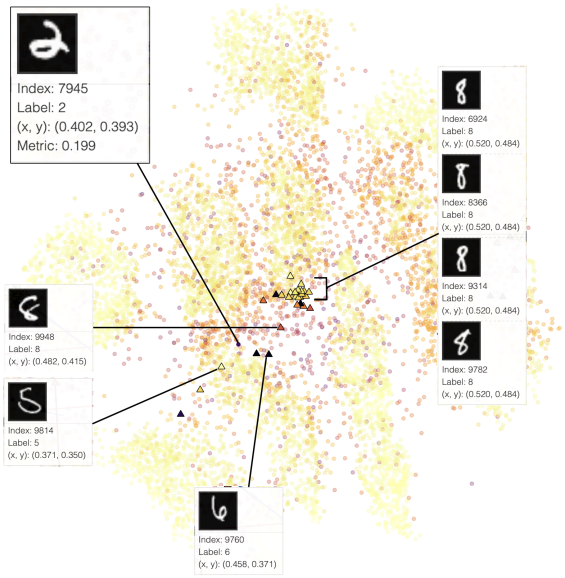


Figure 3.24: Worst point in MNIST Test (7945) outlined and it's 30 nearest training set neighbors by activation.

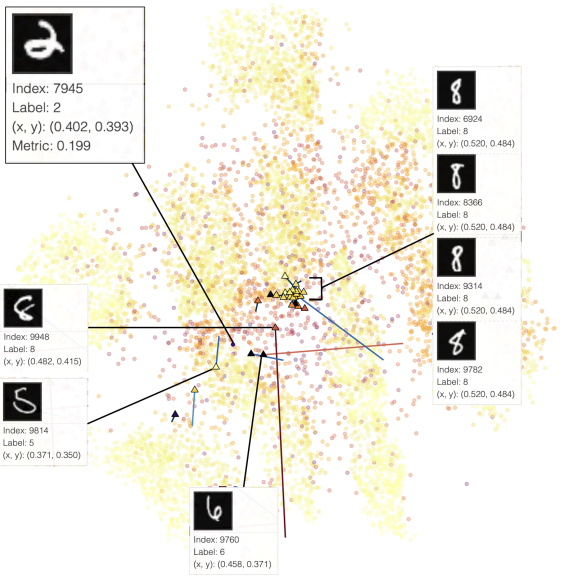


Figure 3.25: Worst point (7945) outlined and it's 30 nearest training set neighbors with interpolation trails showing comparison to t-SNE.

Some property of the input given, which in this case is a direct pixel translation of the picture in the hover tool, leaves the network to learn that these samples are similar. The network's decision process chooses to place this in the middle of the similarity neighborhood. *i.e.* The illustrations of numbers in Figure 3.24 are the pixel input that leads the network to decide these points are similar, and infer its placement at a distance from all similar points.

Is this a Hard Case?

This does not seem to be an unusual looking 2 in any particular way, it's not as angled or distorted as some of the harder cases in MNIST. Visually there is some similarity visible across these neighbors, they all have somewhat similar curved shapes. This point may be far from the learned values because of the variance in neighbors. If this is the case, then we see the same case of repulsion as before, where the network preserves this distance but is forced to balance two groups of similarity.

Examining the Interpolation Trails

In Figure 3.25 we visualize the interpolation trails and things become a bit clearer. As previously discussed, we see a large discrepancy between the choices in t-SNE and NNP. This means that NNP is failing to grasp some feature of this point. This can be due to the distance preservation maintaining the distance from most similar or the learned representations being too far away. Figure 3.26 demonstrates the nearest neighbors in the test set, which are computed by objective Euclidean distance on the test samples. We can see that these neighbors are all over the projection and belong to a variety of different classes. This variety is evidence that the badly performing point is unlike the other points, since it shares some vague feature with points which should have very different features judging by their placement.

Remember that although the neighbors are highlighted by objective distance, the placement of these neighbors is by their learned representation by NNP, meaning that NNP decided to place sixes in the bottom (pink) because it learned those features belong to 6, and the same for 2.

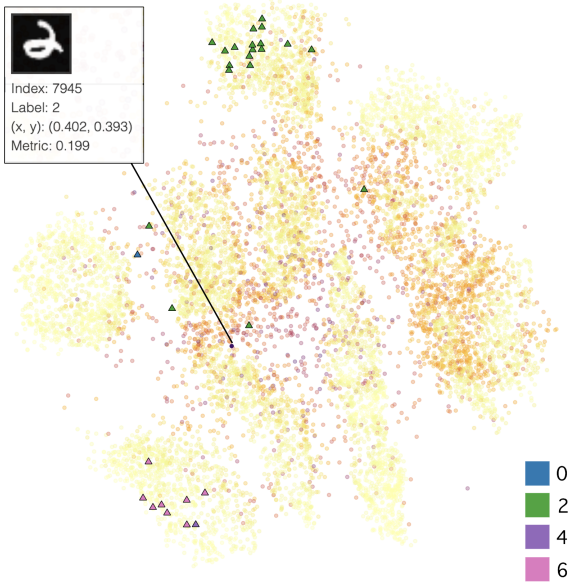


Figure 3.26: Worst point in MNIST Test (7945) outlined and it's 30 nearest test set neighbors by Euclidean distance, colored by class

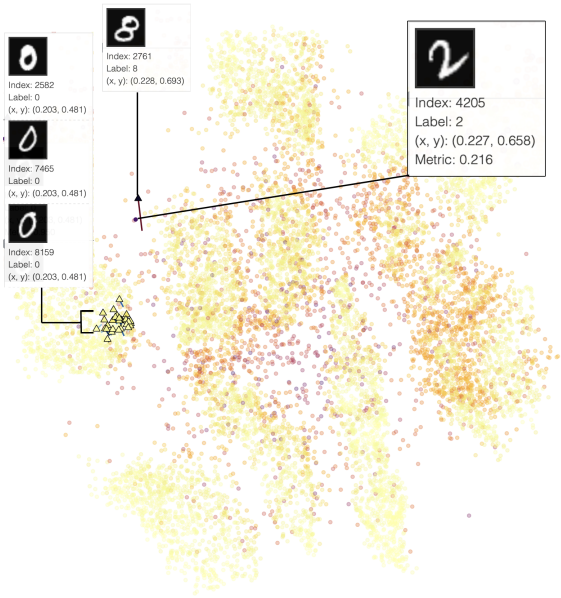


Figure 3.27: The second-worst point in MNIST Test (4205) outlined and it's 30 neighbors with interpolation trails showing comparison to t-SNE.

The neighbors of point 7945 are so spread out because NNP recognises some features of both of these classes and is unsure about where on the spectrum of what it's learned this point should go. This leaves point 7945 as a diffuse point in limbo between the clusters.

The Second-worst Point in MNIST

In Figure 3.27 we see point 4205, the second-worst point in the test set according to our aggregated metric.
This point is a 2 which is placed between a cluster of 0s and an 8 which are considered similar to 4205 by the network.
The interesting thing to note here is that the 8 has a very low metric and appears to be 'corrupting' this point by pulling it from its cluster.
This is an example of the midpoint case from Section 3.3.2.

The trails are not very confused at all. Once again the 8 is the only problematic point, all of the other neighbors are basically agreed between t-SNE and NNP (there are very small trails from the neighbors in the 0 cluster). The activations for the cluster of 0 must be very well formed, since these project very well. 4205 does not seem to be very strange (the shape of the 2) suggesting that this lowest scoring point may not be learned correctly.

We see the test neighbors of this second-worst point in Figure 3.28, and we see a trail of points leading to a cluster of 1s that are objectively more similar according to a distance computation. These points are not placed all around, but rather seem to be pulled by some feature towards the cluster of 1s. It is clear in this case that NNP and the Euclidean distance function applied to D focused on something very different. NNP seems to have focused on the curled shape of the top to differentiate this point, while the objective distance seems to have focused on a the large line of filled in pixels in the center (that are common in ones).

3.5.2 Generalization in Dogs *vs* Cats

The Worst Point in Dogs *vs* Cats

In Figure 3.29, we see this test point (4874) is placed above all of its nearest 30 training examples, which are the nearest as found similar by NNP.

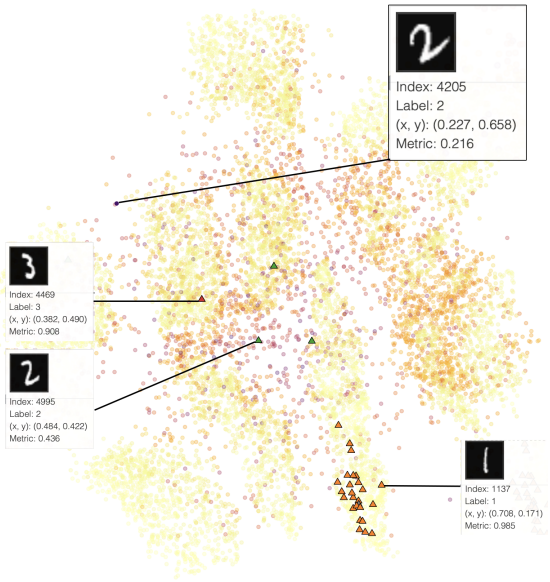


Figure 3.28: The second-worst point in MNIST Test (4205) outlined and it's 30 nearest test set neighbors by Euclidean distance, colored by class.

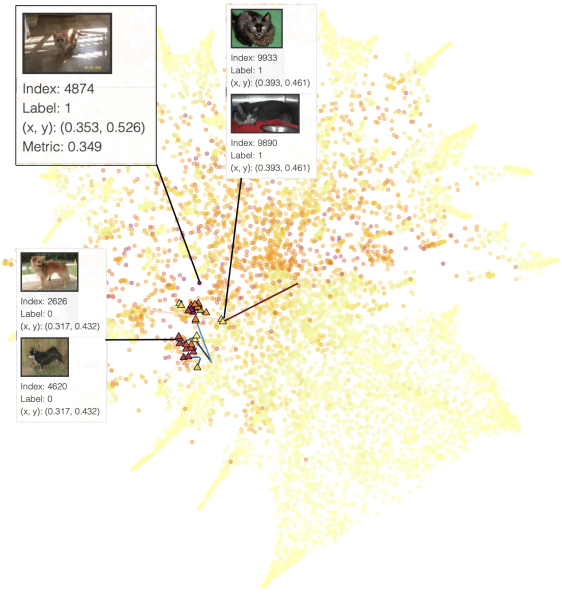


Figure 3.29: The worst point in Dogs *vs* Cats Test (4874) and it's 30 nearest training neighbors with interpolation trails showing comparison to t-SNE.

Given no point pulling 4874 away, it should be predicted as similar to those points NNP learned as similar, however it is not. This is evidence that NNP is trying to preserve some distance between these points even though they are similar in features that it recognizes.

If we examine the interpolation trails in Figure 3.29, the same situation above persists. We do not see points wildly fluctuating, in fact the movement of NNP seems to have clustered these points less closely than t-SNE, and this effect became magnified when inferring based on these points.

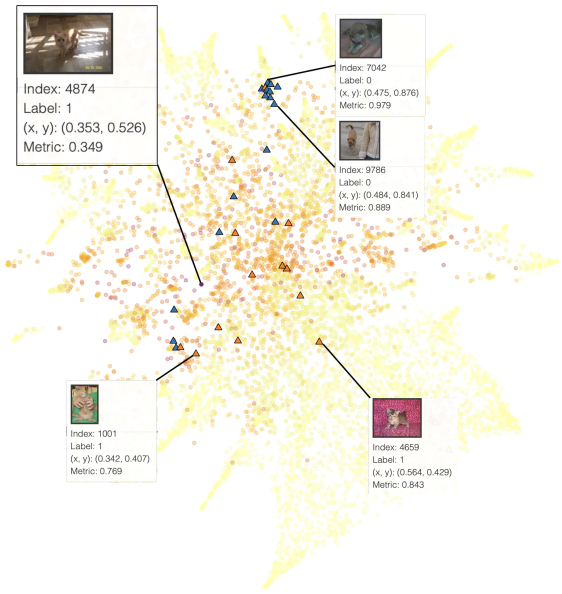


Figure 3.30: The worst point in Dogs *vs* Cats Test (4874) outlined and it's 30 nearest test set neighbors by Euclidean distance, colored by class.

The final step is examining the test neighbors in Figure 3.30. These are spread over class and over the center of the projection. Once again it seems that NNP has learned different features. NNP has decided this is similar to animals standing near sunlight or natural colored backgrounds (brown/green) while Euclidean distance seems to have found all instances of light brown animals across the projection. This point may be far away from the known, as indicated by the spread of the test samples. Since these points do not move much

in t-SNE which is visible in the interpolation we know that NNP chose this placement based on t-SNE. During training t-SNE showed NNP that similar points (according to NNP) belong in this subsection, and NNP may have pushed it further away to be closer to preserve the distance between the t-SNE 'similar' points.

The Second-worst Point in Dogs *vs* Cats

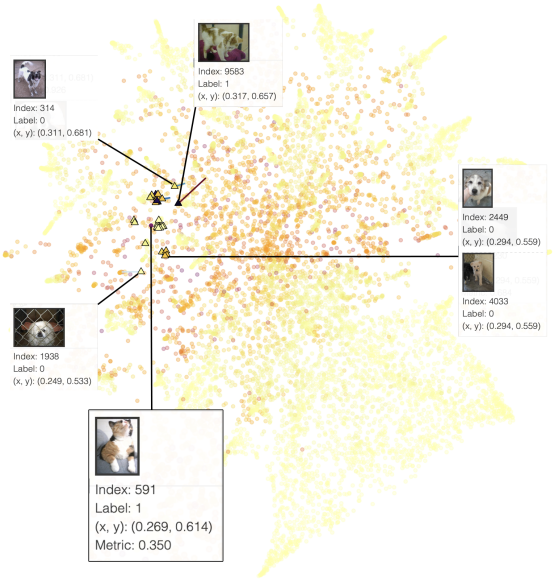


Figure 3.31: The second-worst point in Dogs *vs* Cats Test (591) outlined and it's 30 nearest training neighbors with interpolation trails showing comparison to t-SNE.

In Figure 3.31 we see the second-worst point in Dogs *vs* Cats, at index 591. This seems to be another case of the midpoint theory. All of its nearest activations are nearby and it is placed in the middle. We can see that of these points, only one is dissimilar to t-SNE (9583) as evidenced by the long trail. If we take the lacking of long trails to mean that NNP learned quite well from t-SNE for this point and its neighbors, then we would expect that if this point is similar to those it would be placed well.
Now we look at the nearest test neighbors in Figure 3.32, they are quite spread out. If NNP learned this test point based on some training points placed correctly compared to t-SNE, then we would expect

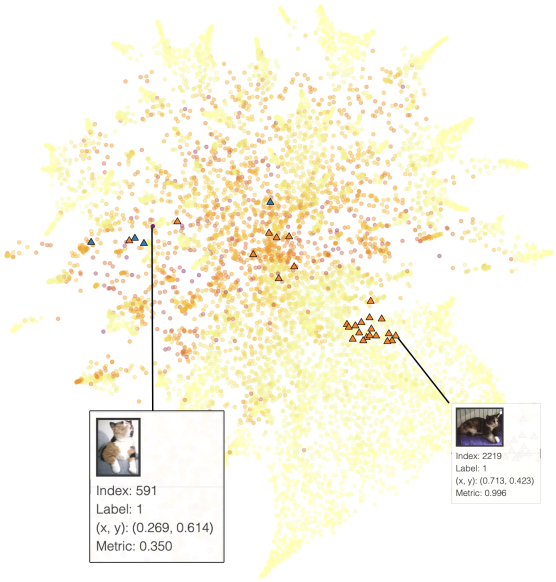


Figure 3.32: The second-worst point in Dogs *vs* Cats Test (591) outlined and it's 30 nearest test set neighbors by Euclidean distance, colored by class.

that it learned correctly. Now that we come to the test set we see diversity in placement again.
The most similar test samples are very different from the most similar training examples, meaning that the testing set was too far away from the known to apply learned representations to this this point.

3.6 Applying our knowledge: Improvement Phase

In the previous section we examined the effect of the input data on the generalization by reasoning about the worst 2 points in each dataset and in every case we reached some conclusion in-line with a theory suggested in Sec.3.4.
We answered the question: *On which points does NNP work poorly?* We conclude that NNP works poorly on difficult or unfamiliar cases, but also when exposed to some aspect of the implementation. What is not clear is the exact causes of this difficulty or failure to learn. We are not sure if there is repulsion, a midpoint or simply distance from the known.

To reach our end goal we need to improve the projection. To do this we need to outline these exact causes. Firstly, if the input data is very far away from the known or difficult, the network may be forced to place it incorrectly by being mislead by similar features it has learned.
We explore further in Sec. 3.6.1 to see if we simply do not have a representative training set, because this is a problem that is difficult to fix. If the input is far from the known: we try to clean up and modify the input data in Sec. 3.6.2, to see if we can remove such misleading components.
If the input data is just difficult, we try in Sec. 3.6.2 to more efficiently learn the more difficult points by reordering the input. There are cases (such as Figure 3.27) where the point does not seem to be unfamiliar, evidenced by the similarity and agreement of the placement of training samples in NNP and t-SNE. For these cases it seems that there is some issue with the projection function of the network, particularly with regard to the network's inability to approximate t-SNE on account of NNP of prioritizing distance preservation over the local structure. We will search for answers to this in Sec. 3.6.5 and try to find a more suitable projection function in Sec. 4.
We have used the knowledge gained from 'Examining the Problem' to outline some theories for the causes of diffusion, namely: distance from the known and distance preservation. We can examine each of these in detail in order to grain understanding and then use this understanding to *improve multidimensional deep learned projections*. First we examine the cause of diffusion we have outlined in the most cases, and with the most certainty, which is a large distance from the learned examples.

3.6.1 Selecting a Cause: Distance from the Known

Many times when exploring the projections in the previous sections, we hypothesized that some points may be too far away from the known to project correctly. It is impossible for a neural network to learn things completely dissimilar to things it has seen in the training set. We have chosen to select and focus on 'Distance from the known' from our list of causes. We evaluate if this cause shows evidence of being a potential source of diffusion and evaluate the extent of its influence.

Isolating a Cause: Is the distance the reason for diffusion?

The evidence we have seen so far brings up the question: *Is distance the reason for diffusion?*. To answer this question, we add a distance tool to our proposed visualization solution. This tool has two visualizations; the first is simply a display of the distance as another heat colormap, while the second computes a division of the error and distance in order to plot a correlation. This metric, which colors the new plot is a function of metric and distance as such:

$$metric = \frac{1 - metric}{distance} \tag{3.2}$$

The intuition for this visualization is that if the distance in the test set is correlated to the error, then we will see a similar color across all the test points.

We have seen a lot of cases which point to distance being a factor, but we are skeptical to say that distance is the only factor.
Doing so ignores the existence of other strange cases such as Figure 3.27 where this point has low metrics despite not seeming (visually to a human) to be an unfamiliar case. This point also has no particularly confused neighbors in training or test sets.

In this section we will justify our hypothesis by demonstrating the distance relation to diffusion. Once we demonstrate that distance is a factor, we will apply the two visualizations of the distance tool on the samples we chose as 'worst performing points' in the previous sections. We hope to figure out if the distance from the known is really the only cause of the diffusion in these worst performing points.

Isolating the Cause

The Figure 3.33(a-d) present the case that the highest distance is in the same places as the highest error and that this is also the location of the diffuse points. The higher we bring the threshold (white points), only focusing on higher percentages of total distance, the fewer dark points we see within clusters and the more they start to align with the diffusion in the projection.

This distance is by activation, so we are judging the distance between test samples from learned representations.

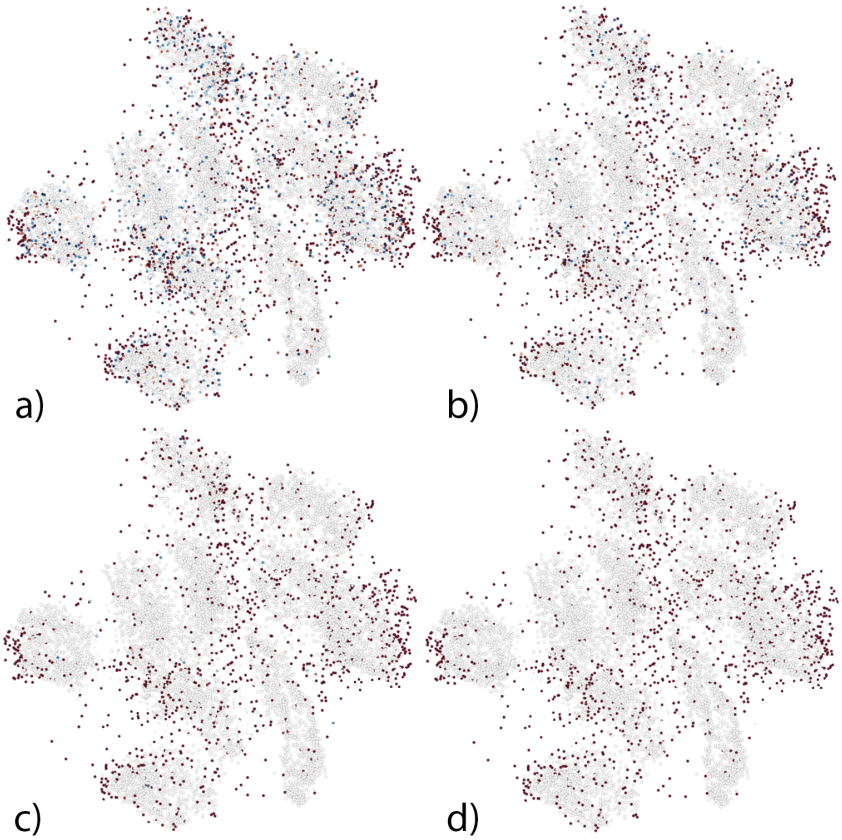


Figure 3.33: Distance colormaps, the highest distances range from blue (relatively lower) to red (relatively higher). White points show a threshold cutoff, meaning distance values below this percentage are not colored. The threshold for each is: a - 0.8, b - 0.9, c - 0.99 , d- 0.999. The higher the threshold, the fewer points within clusters and variance.

This suggests that diffuse points are unfamiliar to what the network learned and as a result, are projected badly. It may be that the network is unsure and is then making very low or very high activations for these points which brings them further from the well projected points. If the points are being influenced this way, it may seem that there are not enough neurons to adequately fit to all of these points. Instead of using a new neuron they signal a certain feature through very high or very low activations. Due to the hyperparameter training done in [3] and the addition of 'xl' architectures during this thesis, we know this is not the case as diffusion persists regardless of additional neurons. The network may lack discriminatory neurons, so that it does not define learned features by which to place into clusters, but rather the combination of activations leads to placement. In which case these distantly placed points hold these learned features much more or much less than other points and as such the network attempts to maintain the distance between them.

Distance Tool on the Worst Test Points

The distance tool works as follows when using the first component, which is simply a heat colormap of distance. The user may apply a filter to threshold the colors, similar to the error threshold, with the same opacity slider applied to it. The color is different in this heat colormap though, as we want to focus on the distance only, and we want to see

the other points for context so that we can identify the diffuse points. So we make any distance above the threshold red and any point below the threshold white. The more red a point gets, the higher it is relative to the distances that are above the threshold.

Distance Distribution in MNIST

In Figure 3.34 we see the MNIST test set, with the worst performing points that we isolated in Sec. 3.5. At that time we hypothesized that they may be placed poorly due to distance; we can now confirm this. The white points are below the distance threshold, while the more red a point is, the more distant it is. Both of the points we outlined as the worst error are very dark red.

By using the distance threshold to slowly increment the threshold, we can determine that point 7945 is in the top 1.4% of distances, while point 4205 is even further away at a distance in the top 0.06%.

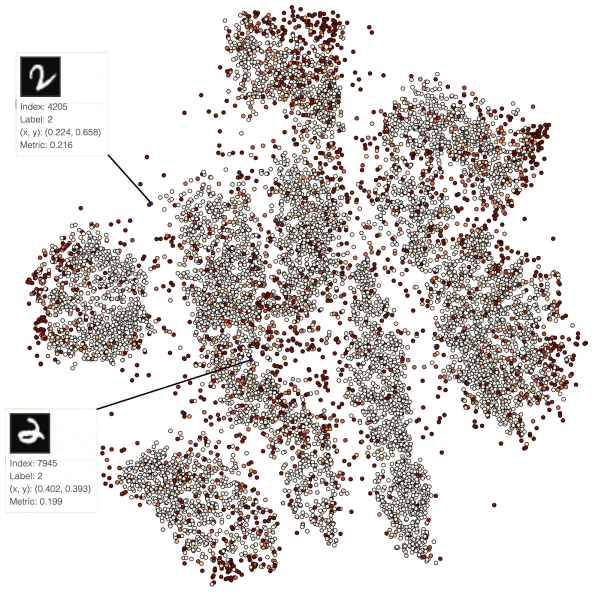


Figure 3.34: MNIST: threshold of the highest distance and the worst test points. White shows points lower in distance than the worst points.

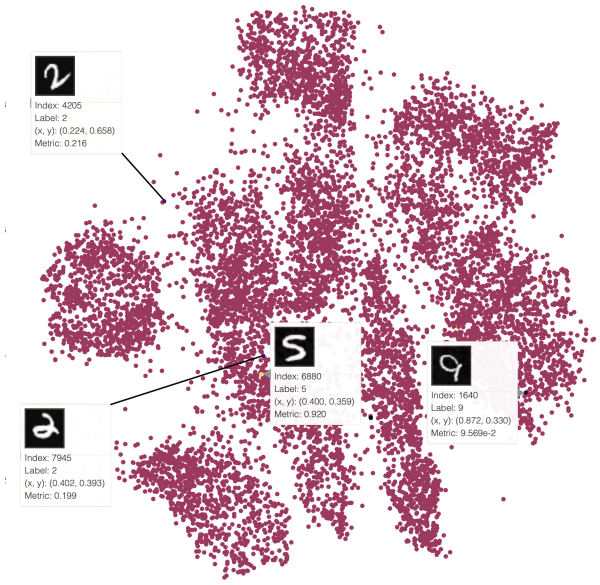


Figure 3.35: MNIST: error/distance, the consistent color shows correlation. The worst test points and two new isolated points which were colored darker or lighter than the surrounding points.

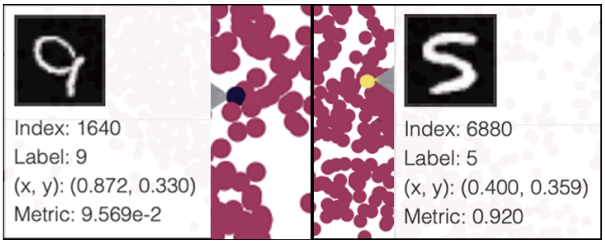


Figure 3.36: A closer view of the lighter and darker points in 3.35

Distance Related to Error in MNIST

Figure 3.35 shows the error divided by the distance, we see a solid color heat-map that confirms that the error is correlated with distance.

There are two exceptions to this case, which are the points 5 (6880) and 9 (1640) added to the graph and shown more specifically in Figure3.36. They appear as very light yellow and very dark black, having a very high and very low divided metric respectively. Remember that the color is scaled according to the metric, so the darkness of the color is relative to the current metric.

Being very dark in this context simply means you are much lower (relatively) than everyone else.

This 5 performs very well in the original data (0.965 aggregate metric), and it has a very small distance so it is very familiar. While the 9 performs adequately (0.754 aggregate metric), and is harshly penalized by its high distance.

This is not a diffuse point, if distance was responsible for diffusion we would expect that it would be, however just because this 9 is harmed by its metric more than the others, does not mean it is in the highest distances. This 9 is within the lowest 3% of points, but Figure 3.33 shows that only the top 1-0.01% of distances contain the diffusion.

The Same Pattern in Dogs *vs* Cats

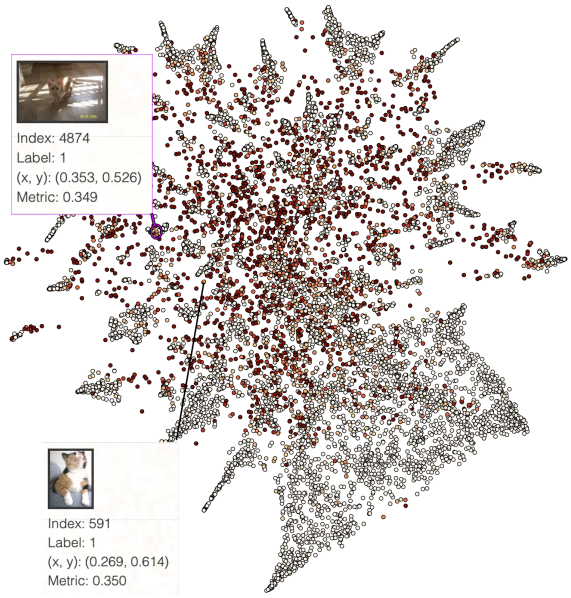


Figure 3.37: Dogs *vs* Cats: threshold of the highest distance and the worst test points. White shows points lower in distance than the worst points.

We repeat the same procedure for the Dogs *vs* Cats dataset and find similar results which are shown in Figure 3.37. The two worst performing points from the test set belong to a subset of the darkest red points in the distance threshold; both are in the top 3% of distances. Furthermore we divide the error by the distance again and find the same pattern present in the MNIST distances, shown in Figure 3.39. We have two kinds of points that stand out, some very light and some very dark, these are illustrated in Figure 3.38.

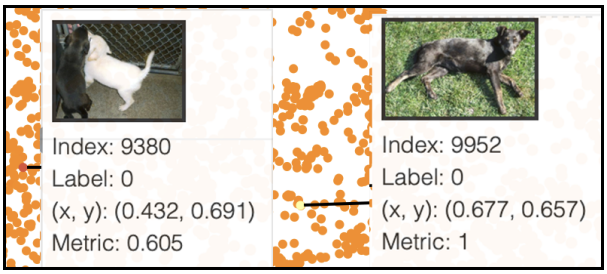


Figure 3.38: A closer view of the lighter and darker points in 3.37

A Cause of Diffusion is Distance

In this chapter we have shown that unfamiliarity, so far called 'distance from the known' is a source of problems in the input and it is also a solid indicator for the diffusion that we want to reduce. We have confirmed this is consistent behavior across both datasets.

Now that we have isolated a cause for the diffusion and applied our tools to understood it, we can attempt to fix it.

The solution to errors which are caused by the distance from the training data may be as simple as adding more representative training examples. Ideally, this would allow NNP learn to better predict similar samples. Extra data is not always available and with the computation time of t-SNE, adding more data every time we do poorly is not a feasible solution. On top of this, the training set will never be completely representative; otherwise there would be no need for generalization and we would somehow have to account for every possibility perfectly. So there is no easy solution for distance in training, however we know that the distance from the known is not the only problem. This means that even if this distance from the known is an irreducible error brought on by distance from training data; we have other opportunities to solve our research question.

So with what we know now, how can we *improve the quality of deep learned multidimensional projections?*

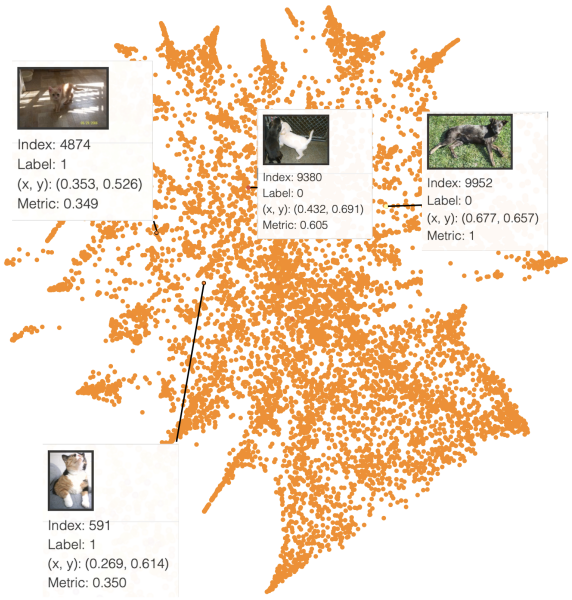


Figure 3.39: Dogs *vs* Cats: error/distance, the consistent color shows correlation. The worst test points and two new isolated points which were colored darker or lighter than the surrounding points.

We cannot completely reduce the error that comes from distance from the training data, but we have found cases which point to other causes that we may be able to solve. Evidence of these causes are the unusually dark or light points in the correlation as well as the worst error not being exactly indicative of the worst distance. We see the highest error in the top percentages of distance, but the highest error is not the highest distance. These two points suggest that distance is not the only problem. In Sec. 3.6.2 we modify our training to try learn easier and clean unrepresentative lessons from the training. Once we have applied these two fixes, we will have outlined and addressed the causes of diffusion within the input and we can move on to examining problems in the architecture. We stop searching for more causes while we implement the fixes to the causes we have outlined, so that we can incrementally improve the model with successful fixes and are not finding additional causes that would be fixed already by a proposed fix to another problem.

3.6.2 Fixing a Cause: Adapting the Input for Distance

So far in the previous experiments, we isolated and focused on some points of error according to our quality metrics and determined that there was some issue that had stemmed from the input data with particular emphasis on distance from the known. Distance from the known may have two causes within the input. The distance we see as error may be created by a failure of the network to learn difficult samples, so we try to teach the network to learn more efficiently so that difficulty is handled better. Secondly, we try to remove points from the training set based on those which perform badly in the training projection, with the theory that these may be misleading or confusing NNP into believing the samples are more distant than they should be. This has the added side-effect of simply removing the highest error, which happens to be some of the highest distance.

This chapter has two parts, one for each kind of input that we use to train NNP, which learns both an input dataset and input projection.

We therefore manipulate the input from both sides to try improve the NNP projection:

- Experimenting with preprocessing and ordering the *raw data* samples
- Cleaning the *training projection*

Fixing Diffusion through Preprocessing and Data Ordering

In this section we try to experiment with preprocessing the input to see if we can improve the network.
The previous paper aiming to improve the model [3] had supplied the model with noisy data, both before and after producing the ground truth t-SNE projection. The idea was to force NNP to learn to be more resilient to noise when placing a point. We then try a similar idea but with different application. Instead of forcing NNP to be more resilient to noise, we try to teach it to be more resilient to the error that we define as low quality metrics.

Firstly, we try to retrain with 'good' data in an attempt to reinforce good behavior. We do this by computing the aggregate metrics on the training projection.

Intuitively the points that are placed badly are the ones that are the most difficult. Then by sorting these metrics in descending order we can get those points that are easy for the network to project and we then refit the network for a few epochs. The result of this is a loss in overall quality metrics. It may be that the good data is easy and refitting to this biases the network towards easy cases, making it lose some of the more difficult niche cases it learned to predict. This line of thought suggests that the network establishes most of its larger weight adaptions when fitting easier points and when learning harder points these weights are adjusted to be detrimental to the easy points, to accommodate the difficult points.

The intuition then, is to try and teach our network as you would teach a human. We start with the easy points, and progressively get harder. The network learns the easier points with less effort and as a result has some context and experience in order to tackle the more difficult points. So we feed the network the points in the order supplied by the sorted metrics in descending order.

We repeat this experiment multiple times to get an idea of consistency. Overall the default order seemed to perform better in most cases, there was no consistent benefit to this training order. We conclude that training with easy points first does not explicitly help the network. We then try the opposite and supply the points in the most difficult first, this also proves to be ineffective.

So there is no explicit fitting to harder points at a later stage of the training and we were just biasing the network weights towards easy points by supplying more data.

We conclude that no preprocessing of the input data in terms of ordering or proportional difficulty seemed to improve the projection.
We now move to try modify the input itself.

Fixing Diffusion through a Clean Input

The neural network can only learn what it has seen. Ideally this would mean that it would have to see a mistake in order to make a mistake. Unfortunately the black box workings of a deep learning model are not so simple. Still, there may be some benefit to not teaching the network some difficult samples. These are samples that may be scoring badly in some metrics in t-SNE such that the network learns 'bad habits' from them. These bad habits come in the form of increased uncertainty in the points that are known, as the weights within the network may be modified to accommodate this new confusing sample.The weights worsen because of this and can no longer perfectly place another sample. We saw evidence of this when learning to fit good points overwrote weights that accommodated strange or difficult points. In this section we try to modify the input explicitly, by cleaning the training projection of bad performing points. We hope this will make the training easier for the network by removing cases that mislead the network into learning undesirable rules and in the end we should get better results.

Removing Points

To explore this idea we remove a subset of the worst performing points, simply through the option to do so on the home page of our proposed visualization solution. Here can select which metric we want to score by, and then how many points we want to remove as a percentage. The visualization solution creates the Ground Truth t-SNE projection, applies the filter on the given metric, removes points up to the specified amount and then trains NNP.

By computing aggregated quality metrics on the t-SNE projection, we can then sort the points by their score in ascending order and simply remove the lowest desired percentage from the training set before recomputing the model. In the following graphs we remove these points ranging from 0%-50% of the total, sorted by highest error and examine the differences in our three quality metrics for each. For comparison on the training set we show the new quality metric subtracting t-SNE, as a comparison. In these cases the default NNP can be seen when 0% of the points are removed. While for the test set, recall that t-SNE is not stable and will change with exposure to new data, making a direct comparison impossible. In this case we subtract the original NNP, with 0% removed. The subtraction in both cases means that we can easily see which is the highest based on how high (green) it is or how much worse (red) it is to the ground truth t-SNE or NNP (white).

Projection on MNIST (10K samples)	NH	C	T
GT t-SNE Training	0.94	0.99	0.98
Clean -22% t-SNE Training	1	0.99	0.99
Default NNP Training	0.89	0.96	0.98
Clean -22% NNP Training	1	0.97	0.99
Default NNP Testing	0.83	0.91	0.97
Clean -22% NNP Testing	0.88	0.91	0.97

Table 3.1: The absolute values of NNP and t-SNE on a default and cleaned training set.

Results of Removing Points
After computing the quality metrics, removing the worst points up to a percentage and projecting this same data as training data, we get the following results in Figures 3.40 and 3.41 for the train and test set respectively. It is important to examine the performance on both training and test set, because the test set allows us to check for two problems that may not show in the training set: Overfitting and Generalizability.

Overfitting
We are cautious of overfitting, since we have fewer samples, in some cases 50% less, but still use the same architecture and train for the same amount of epochs.

Generalizability
We are concerned that regardless of overfitting, we may lose some generalizability just because the network has never seen some points. Recall that we explicitly aimed to show the network less error but struggle with distance cases.

What happens if the network sees a point in the test set that is only similar to 'error' points in the training set?

If we were overfitting we would expect to see an increase in the training set performance, which we do, and if both were occurring we would expect a decrease in the test set performance, which we do not.

NNP is very good at performing with fewer samples, and is able to generalize very well even without a large amount of samples. The graphs show relative metrics, for an easy comparison between the baseline and the removed points, but to further illustrate what the network has learned we show the absolute values in Table 3.1.

By teaching NNP on only clean training data, we can reach metrics much better than t-SNE, almost the same as the t-SNE which had all of its error explicitly removed. We find a Neighborhood hit of 1, which means a perfect separation in clusters. This is a good indicator for performance when competing with t-SNE because of it's preference for cluster separation.

We are also performing better on the test, specifically with Neighborhood hit. There are the same amounts of missing and false neighbors in the test set - remember that the test set remains completely unaltered during this removal, so it may be that any benefit to removed points in the training creates false/missing neighbors in points that the network is unsure where to place because it has not seen similar. We can see that the network has improved however, because the clusters have become better separated even on unseen data, the features that are learned are key to the identity of these samples, at least by label.

Results of Re-adding Removed Points

At this point we have simply removed problematic points, and this seems to be an effective method of improving the model. This is a shallow victory, as we have just forgotten some points rather than improved the process of NNP when dealing with them. So now we perform the same method as before but we pass the entire training set, without removed points, to the network to project. Note that we did not move these 'removed' training points into the test set, we simply supplied the original training. This means that the training projection is evaluated on this full training set, which now contains unseen points which must be generalized by the network and may score lower than expected because of this. The results of this are shown in Figure 3.43.

The visual effect on the projections is displayed below in Figure 3.42. Recall that t-SNE is unstable meaning that subsequent recreations of the t-SNE projections with different points leads to different layouts, hence the differing projections.

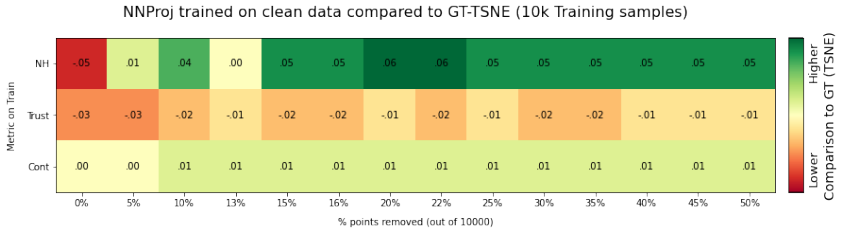


Figure 3.40: The quality metrics after cleaning the projection MNIST train projection. Darker green are higher than t-SNE, red are lower.

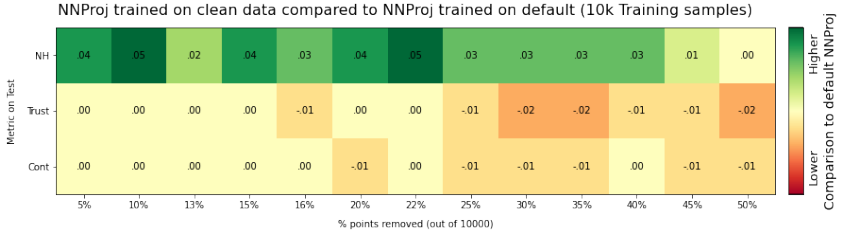


Figure 3.41: The quality metrics after cleaning the projection MNIST test projection. Darker green are higher than default NNP, red are lower.

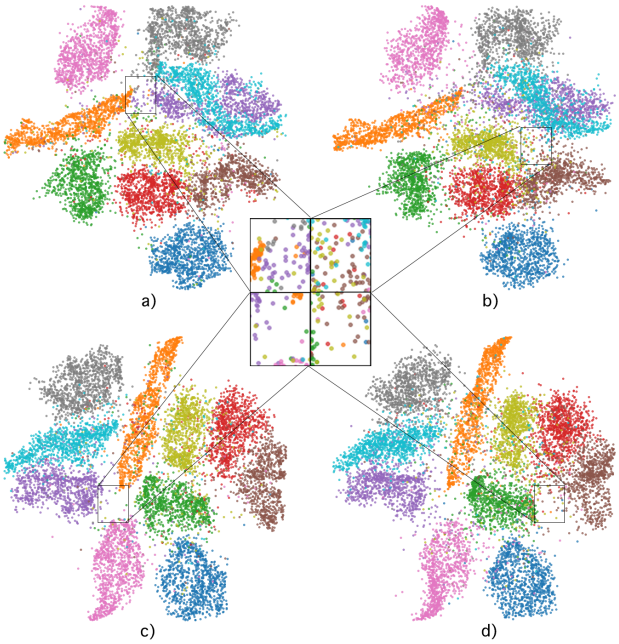


Figure 3.42: The comparison of diffusion on NNP projection of MNIST when training with removing 0 points removed (default) on the training set(a), the test set (b) and with 13% removed, on the training set (c) and the test set (d). Notice the seemingly less diffuse points between clusters.

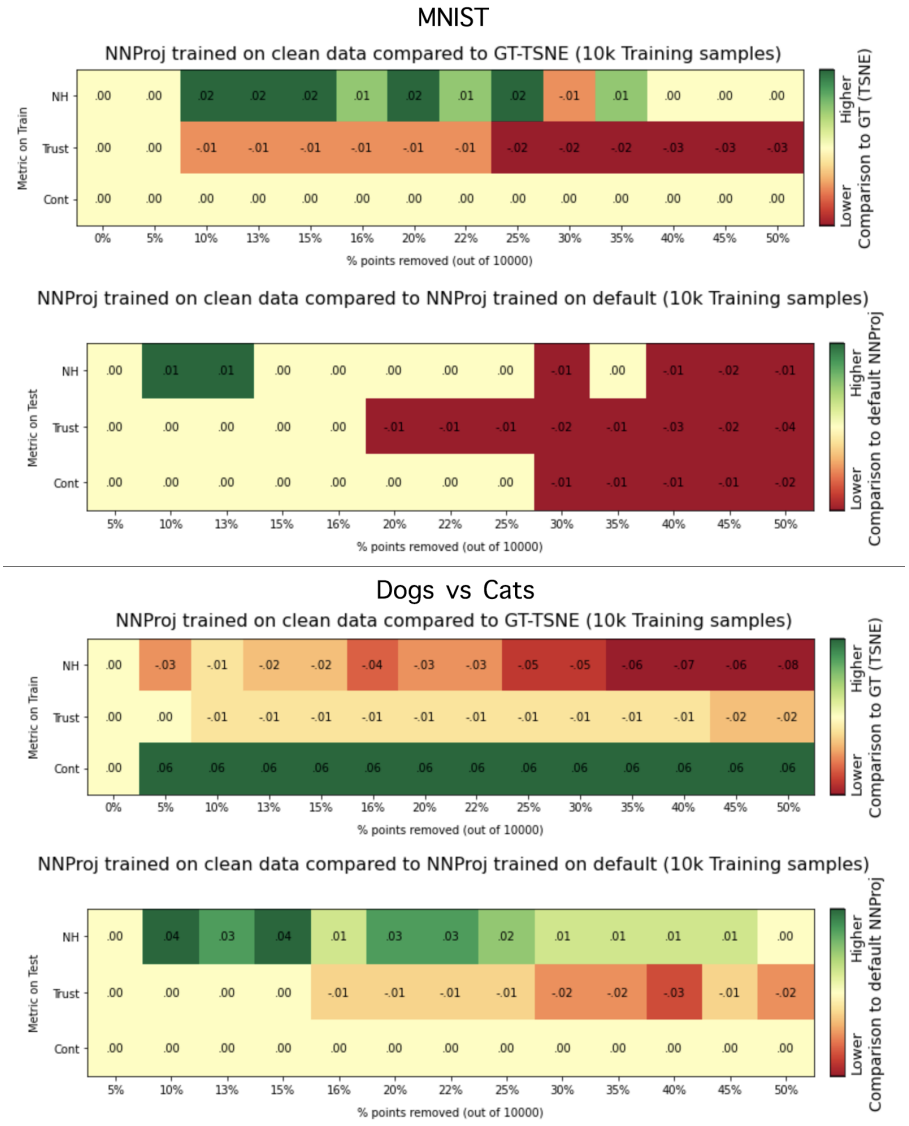


Figure 3.43: The quality metrics after cleaning the projection MNIST test projection and adding those points out-of-sample. Darker green are higher than default NNP, red are lower.

3.6.3 Re-evaluating: Distance from the Known

After we have applied a fix, we need to recompute and re-evaluate the performance of the solution. In this case we found that the final step did not yield an overall better model than before the fixes were applied. So we revert back to the default version. The most important re-evaluation was at the time of computing Figure 3.40. At this time we had removed some badly performing points from training and had not yet re-added them to the metrics. We examine the same steps as before in the workflow (Figure 3.1), we recreated NNP so we enter the 'Examining the problem' phase again. We find that when examining the training set with the removed points compared to the default version without removed points, the variance in the range of distance is drastically reduced.

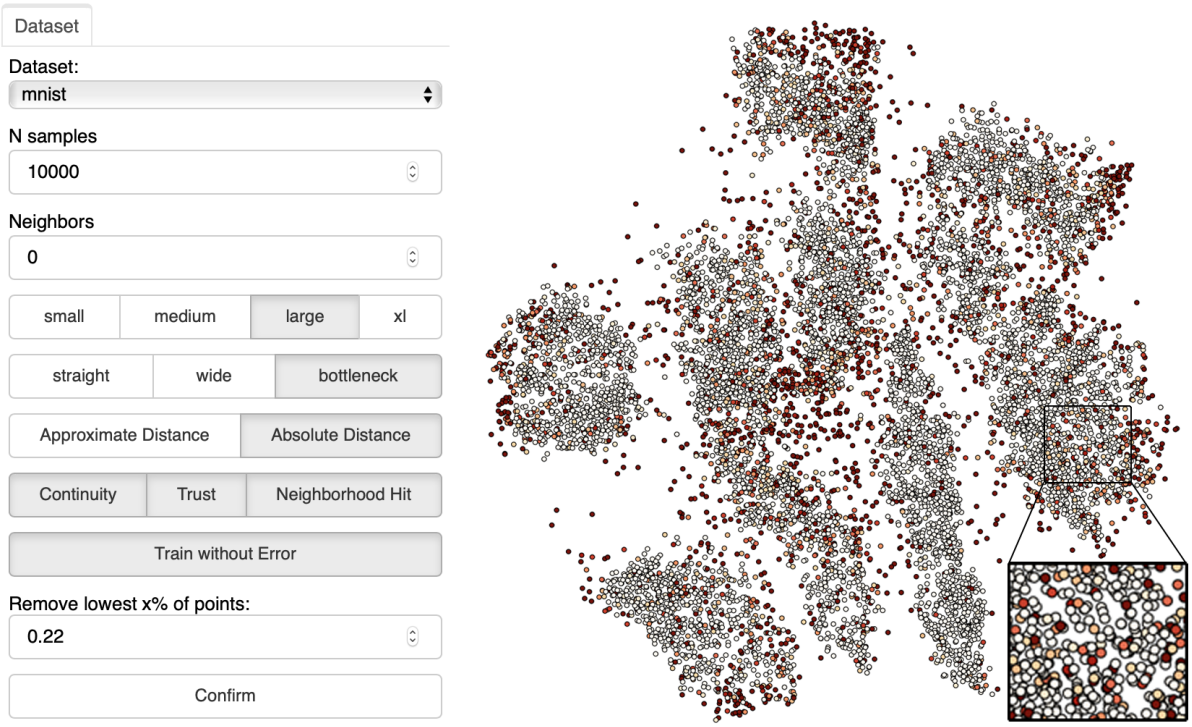


Figure 3.44: The distance on a clean projection, insets show inner cluster distance

We observe this through the distance tool, which we used in Sec. 3.6.1. We found that there was some relation between distance and error, but that distance was likely not the sole cause of diffusion. The ranking of distance did not match the ranking of the error we relate to diffusion.
The important thing to note here is that we say we reduced the distance range, not that we removed the distance. We removed the highest error points, in Figure 3.44, which left distance within the clusters themselves. These points are not obviously diffuse, nor are they error points. This contrasts the distance we see portrayed in the figures in Sec. 3.6.1. This is important to bring up at this stage, because it confirms that the distance is not the only factor that causes diffusion and we cannot have some quick fix based only on the distance, nor can we give up finding solutions based on irreducible error as a cause.

3.6.4 Selecting a Cause: Distance Preservation

Previously, we tried two methods of correcting potential causes in the input. First we changed the order of the input and found that it didn't improve the projection significantly. Then we tried to provide a clean projection example for training.
Although we have found some improvements we lack a fundamental change that makes NNP better. The other cause that we had outlined previously during the 'Examining the problem' phase is that of Distance preservation, that some property of the architecture of NNP is preventing the points from being placed closely together. Now we move our focus to narrowing down on that cause.

3.6.5 Isolating a Cause: Neurons Responsible for Patterns

In this section we will examine the activations of groups of similar neurons in order to try and find some property in activations that distinguishes the well clustered points from the diffusion.

We want to be able to tell which parts of the NNP architecture are responsible for which parts of the projection by examining the activations.
If we can isolate a subset of the neurons which activate predominantly on diffuse points but not on the well-performing points, then these may be contributing to the diffusion. We may then be able to modify the weights to those neurons to make them less significant and therefore lessen the diffusion.

The Activation Tool

In this section we will answer the question: *Which parts of the network are responsible for which parts of the projection?* To answer this question, we add an activations tool to our proposed visualization solution. We get these activations through the same method of extraction in Sec.3.3.3 for the nearest neighbor tool.
We create a surrogate model without the final activation and extract the activations by predicting with this model. This idea can be generalized to get the activations from each layer of the network. Recall that NNP uses very few layers so it is not difficult to retrieve all of them.
To define the query, the user selects some points. The user invokes the activation viewer by simply toggling a button. The activation graph of each selected point is then added to the interface.

Earlier iterations of the Activation Tool
Originally the activation graphs depicted the average of neuron activations over all of the points, with the logic being that averaging them would smooth out the differences while keeping the definitive points across groups.
In contrast to smoothed commonalities these definitive points would be more obvious. What we found instead was that this smoothed all of the activations and no obvious group was defined for different subsections. This may have more to do with actually lacking different subsections than an issue with display, but we wanted more detail. We then changed from this averaging view to one graph per point, the initial idea being to display this graph as a compact line of tiny neurons per layer for each layer in the network, where each neuron is a circle with a color corresponding to the strength of the activation. Finally we settled on an implementation of individual bar charts per layer. This allows the color and size of bars to convey activation strength while keeping each node unique. This approach has the disadvantage of isolating every activation and making comparing them difficult. To assist with this we link the bar charts, so that using the zoom tool, or panning on the graph prompts the same actions in all the graphs. For a user this means that if you are examining node 0 in the input layer of the first selected point's activation, all of the other selected points will focus their activation graph on that node. In this way every graph shows the same position to allow for easy visual comparison. The process becomes as simple as trying to find a node of interest and zooming in on it to compare in all selected points. The hover tool can be used on these graphs as well, to convey the exact activation value and node number.

Adaptability
The activation graphs build and scale on screen based on the amount of neurons in the currently active size and style of NNP. The first layer at the bottom of the bar charts in Figure 3.45 corresponds to the input layer, the second is the second hidden layer. In this case this is the bottleneck layer (recall we use the best implementation from [3] which is a large bottleneck) making it smaller than the other layers. The third line of bars from the bottom is the 3rd hidden layer and the top-most line of bars in the bar-chart is the sub-final activation.
The activations are automatically computed for all selected points, as well as any nearest neighbors placed by the nearest neighbor tool.

Examining Activations
To understand which parts of the NNP architecture are responsible for which parts of the projection, we can examine activations according to three subquestions which can give us understanding into the parts we want to differentiate:

- 1) are close/clustered points affected by small groups of neurons?
- 2) are there different activation patterns when examining the inside and borders of clusters?
- 3) is there a obvious discriminatory pattern in diffuse points compared to clustered points?

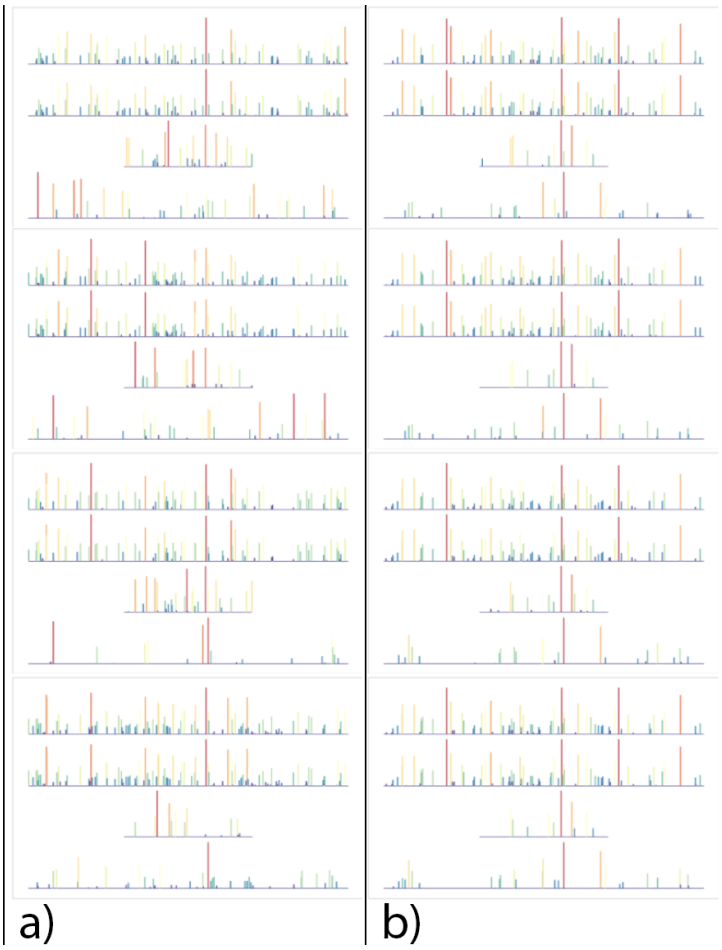


Figure 3.45: Activations of the two groups we wish to differentiate, clustered (a) and diffuse (b). Differences are not obvious.

Close Points and Close Neurons

We perform the examination of these activations on large groups of points; we choose to examine 50 at a time. Figure 3.45 shows a subset of the examined points. Are close points affected by small groups of neurons? Yes. When comparing the close points and the diffuse points, we see much more focused activation in small groups in the network, however this answer generalizes to: diffuse points are affected by a larger amount of small groups of neurons. Rather than isolating the close points, we see their behavior is quite similar.

Differences in the Inside and Borders of Clusters

Are there different activation patterns when examining the inside and borders of clusters? Yes.
We see three different kinds of patterns in Figure 3.47, one for each: border, inside of cluster and diffusion. Although what we do not expect is that the cluster borders have a stronger diffusion in terms of activation than the diffuse points. We see an average of the most varied 'hot' points in the borders of clusters, while the most focused 'hot' points are within the clusters and the diffuse points are in-between.

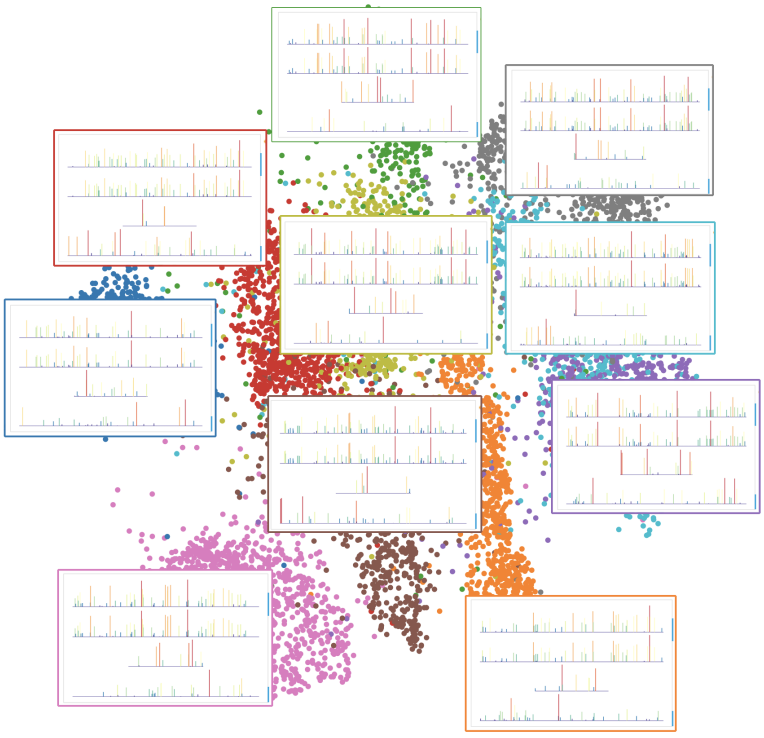


Figure 3.46: Internal cluster activations across locations in the projection

Are there Differences in Activations across Parts of the Projection?

We are unable to find groups of neurons that explicitly differentiate diffuse points, but we know from this experiment that the internal cluster points have very focused activations. Do these activations correspond to a particular placement? In Figure 3.46 we take a point central to each cluster, and examine its activations. We may expect that because we compute a regression problem that the lower values in x and y would have lower activations. This may be the case, but it is not obvious. There are fewer high activations in the bottom graphs. Overall these lower graphs do not seem particularly low, or different from other groups. Furthermore we do not see a distinction between left and right parts of the projection in any layer of the activation graph. This tells us that the decision process is more complicated than an activation of certain groups of nodes to reach spaces in the projection, which further evidences the lack of discriminatory neurons.

Lacking Discriminatory Neurons

We had hoped to isolate some component that was responsible for diffusion, which we could simply modify or remove from the network to stop the diffusion, but we were not able to identify such a component. Perhaps because the network is fully connected, or because it is a regression problem we find patterns which we can hypothesize belong to certain points but we have to infer these to a cause based on the neural network's broad architecture rather than a focus point within activations.

As evidenced by the results of Figure 3.47 where activations seem to be superlatives of each other, it would seem that diffusion is not caused explicitly by a property of the network, but rather a side effect of what is happening to the points at the borders applied in a less strict manner. We can see this through the diffuse points being less severe 'hot' areas when compared to the cluster borders. This means we cannot target a focus point in the layers.

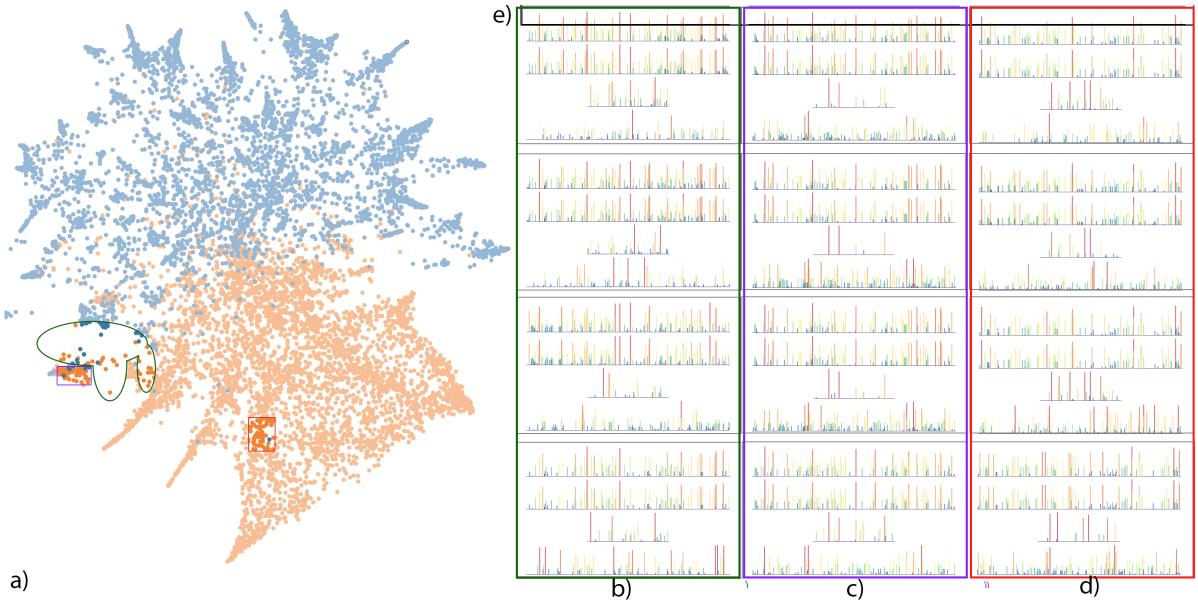


Figure 3.47: Groups of neurons examined in Dogs *vs* Cats, their placement shown in (a). Diffuse (b) with an average of 27 'hot' neurons, black box (e) as a rule to count the 'hot' in the top row. The distant cluster (b) with an average of 30 'hot' neurons and an internal cluster (d) with an average of 7 'hot' neurons.

We are sure that this activation difference is a property of the distance preservation but have seen multiple manifestations of this. We have two cases, either the activations at the borders are more similar to those points outside which makes the network strictly focus on both to preserve the distances, or we see the distance preservation based on the similarity without the comparison to another dissimilar point.

3.6.6 Isolating a Cause: Midpoint or Repulsion

In the previous section we came to the conclusion that the activations are diluted because the network is explicitly trying to differentiate points as a product of its implementation. We also concluded that this differentiation seems to specialize at the borders and clusters. So far we have suggested two possible cases when we came across indications of the kind of problems that relate to this differentiation.

Firstly in cases such as Figure 3.27, the point became diffuse by seemingly being pulled away in placement by sharing learned features with a distantly placed point. In such case the network prioritizes the shared learned features with a distance point, rather than similarities with otherwise nearby points. We hypothesize that this situation is a product of differentiation between multiple groups of similar points and as a side effect our diffuse point is placed at a midpoint of similar subsets of neighborhoods.
This is our *'midpoint'* theory.
Secondly, in examples such as Figure 3.22 we see a point that does not seem to be compared to any particular distant features, but is pushed away from its most similar points anyway.
This is our *'repulsion'* theory.
In this section we examine some such points, to go deeper into the two theories and attempt to understand one root cause for both of them. We have already examined the worst performing points in all views, so now we will simply choose some specific diffuse points.

We will examine these cases, and their nearest neighbors to try and find if a distantly placed point with similar learned features is causing each of these points to become confused, or more likely: that the network is distancing these points because it wants to the preserve distance it sees in dissimilarity.
If we examine the neighbors and find a corrupting influence in all cases, then we know that the midpoint scenario is the cause we see. If we find a case where we do not find a corrupting influence we know it is the repulsion case, since the midpoint requires a corrupting influence.

The Context of these Points

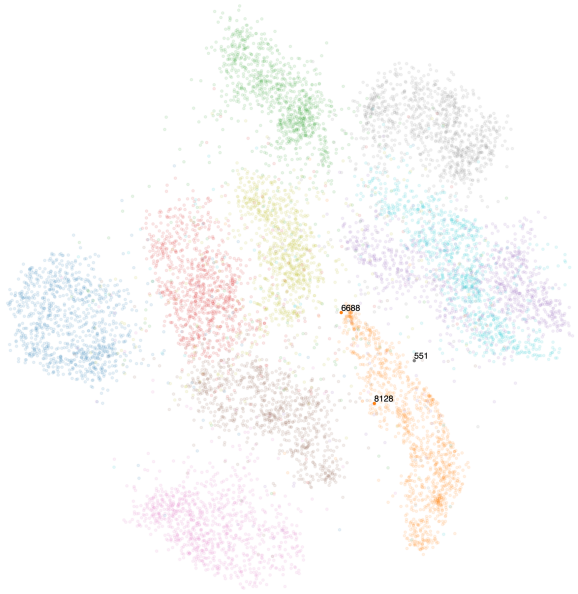


Figure 3.48: Three diffuse points, 6688, 551 and 8128

We have found these three points, (Fig. 3.48) originally chosen in order to investigate if diffusion was somehow learned in the test set, by being similar to diffuse training points.
The process of this was to select groups of test points of greater than 10 within the diffuse points and examine their nearest training samples.
After examining a few such groups we concluded that a diffuse test point does not often have many diffuse training points and a well performing test point can easily have many diffuse training samples to learn from. It is obvious that the points do not learn 'diffusion' as they would a class label, but from this we can also know that there is no property on the input that once learned leads to propagation of diffusion into inference.
This means that diffuse points are learned as placement compared to what the network has learned, happening to end up diffuse as a side effect of the computation of 'known'. This is evidence that there is some underlying structure in the network that conforms to this rule, and that is where our problem lies.

Distance Preservation

So we are here to show that the neural network's function wants to preserve the distance and that, as a side effect causes diffusion. We know that the network wants to preserve distance between the points, in contrast to t-SNE which preserves neighborhoods.
However, we do not know if the distance being preserved to cause diffusion is exclusively distance between several points or distance equivalent to similarity between at least one point.
We want to demonstrate that these diffuse points, and their neighborhoods indicate that distance preservation that specifically pushes similarity away is the problem rather than a midpoint between outliers.

The Neighborhoods of Point 6688

First we begin by evaluating point 6688. This point does have distantly placed neighbors, the first appears at 23 neighbors and more arrive after that. These neighbors are shown in Fig. 3.49. At this point, it seems as if we do have the midpoint situation here. However the point 6688 is placed much closer to the cluster within the orange than we would usually find with the midpoint case; it is not actually at a midpoint. When examining a typical midpoint case we found that the neighbors did not seem to matter in quantity or ranking to pull the point away. It seems then, that this point was placed near the cluster because it was more similar to that cluster, which is more in line with the *repulsion* case.

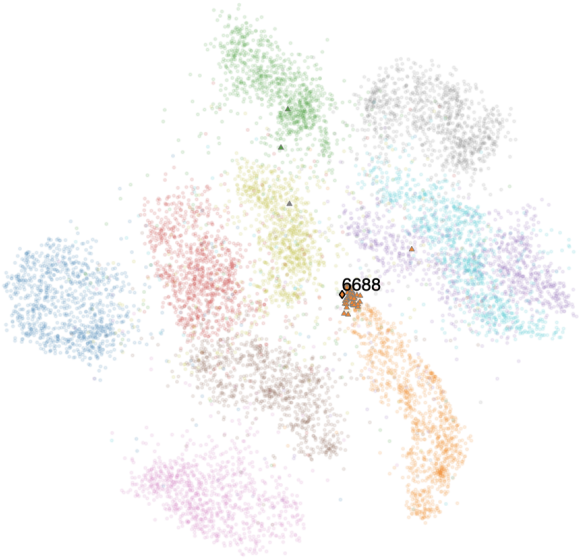


Figure 3.49: Point 6688 and its 60 nearest neighbors

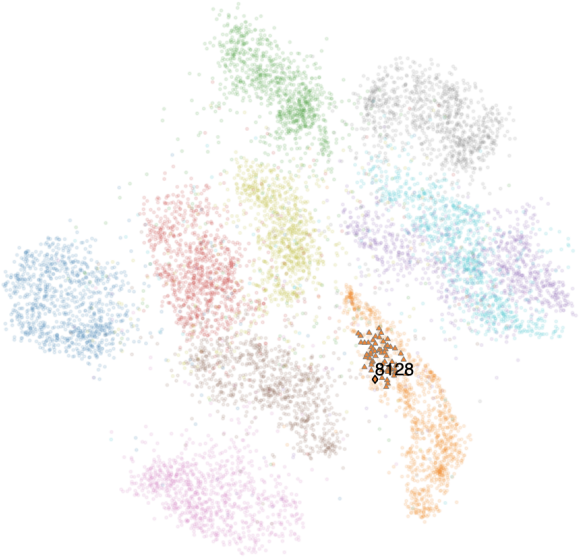


Figure 3.50: Point 8128 and its 60 nearest neighbors

The Neighborhoods of Points 8128 and 551

In Figures 3.50-3.51, we see the same pattern across these two points. We have a diffuse point and its closest neighbors are all within a cluster nearby.

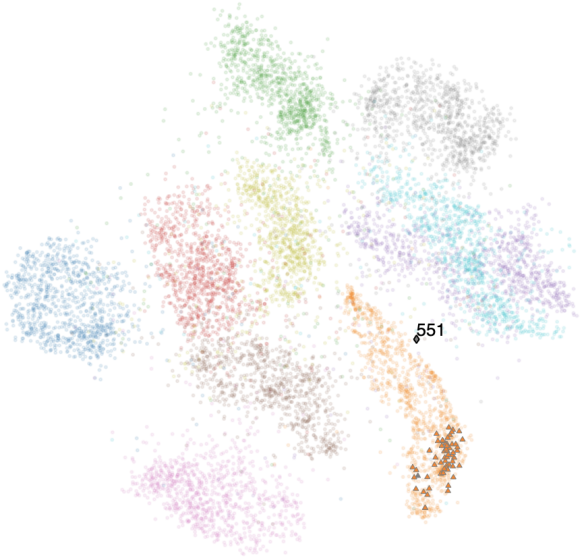


Figure 3.51: Point 551 and its 60 nearest neighbors

There is no consideration of a midpoint when placing these points, and we cannot seem to find a distantly placed neighbor in the most similar points. This distant neighbor may be at a similarity that's incredibly low, but at that stage we would hope that it doesn't influence the projection as much. We can take this result to mean that the network is trying to preserve distance, by pushing these points outwards away from the others and not by placing exclusively between points.

Re-evaluating: Distance Preservation

We examined the activations of the points and found no discriminatory groups. We did find some patterns that suggest that the network is more dispersed in activations when placing points on borders and the diffuse points. We were unsure if points were pushed away due to a distantly placed outlier which was pulling the point towards it, or the network trying to preserve distance from the few similar samples and this happened to push the point away. We are reasonably sure it is the second case, but in either case the problem relates to distance preservation and the network's architecture, so it's time to try and apply a fix to that in the next chapter to finally accomplish our research goal of *improving the quality of deep learned multidimensional projections.*

3.7 Discussion

In this chapter we discuss the performance of this thesis in context with the research goals we specified in Sec. 1.

We set out to *improve the quality of deep learned multidimensional projections.* To reach this goal we split it into two smaller questions: *'How can we understand the diffusion or limited quality in projections?'* and *'How can we improve the projections once we have this understanding?'.*

To examine the first question we split it yet again into it's most atomic parts, each representing the components of the network left unexplored before now. These atomic parts are: *'How can we understand the diffusion or limited quality in projection as a product of the input?'* and *'How can we understand the diffusion or limited quality in projection as a product of the architecture?'*

3.7.1 Find and Isolate the Error

To be able to understand or find the causes of diffusion or limited quality, we first had to define what that means. In Sec. 3.2 we applied the tools we introduced in previous sections to outline what we mean by error. We described the issue with trying to relate the classical kinds of error to our problem and successfully isolated the error we wish to reduce through a combination of the quality metrics that were used to assess NNP in the original work.

We showed that NNP captures the structure and error of t-SNE, such that a reduction in error will guarantee proximity to t-SNE.

We then introduced tools that isolated these points of error, for ease of use in focusing on the highest error and demonstrated the ability to isolate these points in a range of ways.

Contradictions with Existing Studies

We did not find hot-spots in metrics similar to those in the literature, particularly in [21]: the false neighbors were found at the centers of clusters. Both the method used there (LAMP) and t-SNE prioritize local clusters so we would expect to see similar patterns. The reason proposed in the paper above is that the projections struggle to place points within the limited space within clusters and thus we get false neighbors. We reason that the difference could be due to the fact t-SNE and therefore NNP, do not struggle with this because t-SNE is given a lot of leeway due it's t-tail distribution when placing points which are not in local structure.

3.7.2 Exploring Causes

In Sec. 3.3.2 we applied all of the tools so far in an attempt to answer: *How can we understand the diffusion or limited quality in projection?.* We hypothesize causes to understand diffusion related to either the input or the architecture.
We outlined a few hypothesis, theorizing that the main factors that cause diffusion are some combination of: Distance from the known, repulsion by distance preservation and the placement of a point at a midpoint as a method of preserving distance between a cluster and a distantly placed point with similar learned features.

We reached these hypothesis through a demonstration of visual representations that show the relationship between observations and the predicted outcome using the nearest neighbors tool, which is fundamental in gaining understanding into the black box network.

Defining Sample relationship by Weights

Ideally we would like this visual representation within the neighbors tool to show a weighted contribution for each test point based on the training points but we have already discussed how we found that neighbors do not contribute more depending on proximity or quantity. Furthermore it does not seem useful to compute the weights separately, as these are factored into the activations and also change with every training epoch, making it impossible to pass in a sample and check the changes in weights attributed to that sample. This would make the order of training very important as the first few samples would have much larger contribution to shaping the network from its base structure. This assumption is based on the findings of [14], that there was some structure in activations before training but the structure after training was much more specialized and completely overhauled in visual separation.

3.7.3 Narrowing Down a Cause: Distance

We explore the first cause, 'distance from the known' in Sec. 3.6.1. We demonstrate tools to view the projections from various perspectives related to distance. The results show that there is a strong correlation between distance and error. However we also find contradicting exceptions to this in one or two points. This is in-line with the hypothesis that the causes of diffusion are not only those found in the input, meaning they cannot be fixed by modifying the distance alone. Despite this we show that distance is a good measure for the location of the diffusion. With the findings that the reason for distance from the known is tied to the activation spread, along with the theory that diffusion is a side effect of distance preservation, it suggests that the hotspots of distance within the clusters are also diffusion. This would be the distance which remains in clusters after cleaning. This diffusion is not the kind that harms the quality metrics so we did not see it. It appears as a slight misplacement within a cluster so it is obscured.

3.7.4 Fixes for Input Problems

We showed in Sec.3.6.1 that the distance has some kind of relationship to the error. To use this understanding to improve the projections we try two approaches:
Based on the application of noise in [3] to make the network more resilient, we tried to give the network easier samples or bias it, but we found that the order of points did not matter and neither of these helped with the quality metrics. This makes some sense as the weights adapt after every batch of training samples, with no knowledge of what makes then difficult or not. It is more likely that the training weights do not adjust to specific, difficult or unusual cases, rather they learn a general rule of well performing cases and apply that to the other points. That is why we see the difficulty in placement, because they are difficult as a result of general rules not specific ones.

In Sec. 3.6.2 we try to apply a filter to grab some 'low hanging fruit' that is, to expose the network to less error in the hopes that it will learn less error. This works, but if we re-add the removed points this does not have quite as impressive results. In examining this 'clean' data we perform another experiment to examine the distance on this new projection. The highest distance points in this clean projection are not all diffuse, nor are they all error points. The results confirm that the distance is not the only reason for the error, evidenced by the in-cluster distribution of the highest distances that remain in the projection once the most error is removed. As a result of this, distance remains a good indicator of the diffusion but we are sure it is not the only cause.

3.7.5 Narrowing Down a Cause: Activations

In Sec.3.6.5 we explore the two other causes that we have named 'midpoint' and 'repulsion' with the intuition that they appear from the same root cause. We examine the activations in different parts of the projection, finding that the points within the cluster have the most focused activations.
Both the points on the borders of clusters and the diffusion have high average activation, with the borders having even more spread in activations across neurons than the diffusion.
These results suggest that the focus found in these activations may be a part of cluster identity, while the spread activations explicitly push points further from this identity. In this case diffusion would be those points pushed away, but less explicitly because they lack core identity in the first place.

To reinforce this idea of cluster identity we notice that in the input layer the clustered points react more strongly which may give NNP a stronger idea of where to place the points immediately. This strong reaction is a strong identity. The activation layers are less specified in the diffuse points, leaving them more open to identity defined by a distantly placed point with similar learned features. In cases such as Figure 3.27, the activation is disproportionately split between a distantly point and other neighbors which are focused in one group. Alternatively phrased, these diffuse activations are not as close to the others because they have some property learned in a distant point that makes them stranger than their neighbors and the network is deciding to distance them because of that distance to the cluster identity.

Differences from related work

The lack of discriminatory neurons differs from the existing literature in related works, such as [14]: the last layer of a classification network had become increasingly proficient at discriminating groups closer to the output layer, with each later after input refining the cluster coherence. NNP does not behave like this, which can be explained as a product of the classification vs regression difference in the application. If classification decides on a class, and regression computes a number then it makes sense that our network is not as decisive in cluster separation. This may suggest that regression approximates a simple function. We briefly explored if this was the case by observing if there was some explainable decision present in the activations from the core of each cluster. We expected to see higher activations relating to being placed higher in the projection or the opposite and concluded that there was no obvious computation of such. So far in this thesis we have seen no qualitative or quantitative property related only to diffuse points, which is in-line with the hypothesis that these diffuse points are a side-effect of the distance preservation rather than caused explicitly.

We had wanted to examine activations to find some layer or group of discriminatory neurons, that we could tweak to improve the network but no such easy fix was found.

Re-examining Distance with Understanding

The understanding of the activations gives us more insight into the distance problem in the previous sections: The distance in activations went down when we removed the error points, this is because the activations that belong to the error points are very large and very spread while those that are not have focused, relatively smaller total activations.

3.7.6 Narrowing Down a Cause: Midpoint or Repulsion

In Sec. 3.6.6 we go deeper into the two causes which we hypothesize are one and the same: midpoint and repulsion. We try to eliminate the behavior in the two cases above which does not fit, so that we can isolate the true cause. We examine points which seemingly have no distant similarities which could cause the 'midpoint' scenario and therefore reason that the second outcome, that the distance is preserved by repulsion is more likely. We can use these results to then justify the appearance of the 'midpoint' case by re-framing it as repulsion from both sides.

If NNP is only concerned with preserving distance to similarity then the diffusion has nothing to do with the learning of dissimilar points. Rather the network wants to preserve the distance from the existing points. It does not matter if there is another point, the network will place far based on distance from cluster samples. In the cases where we do have a similar point that is far away we are placed at a midpoint of the similarities because of repulsing distance from both sides rather than a computed midpoint. This leads to placements which are not biased by rank or quantity. This mechanism makes sense given the cases we have seen within the diffuse train and test samples, as well as the layout of distance in the projection relating to activation variance. The lack of relation diffuse points have with each other in terms of training samples or qualitative properties can also be explained as the points having nothing in common but happening to end up near each other as a product of repulsion.

Learning from a Neighbor

The inability to find an distant point in the two cases outlined in Sec.3.6.6 does not rule out the possibility that one exists. The distance in similarity that one would have to be to not be featured in the most similar 60 points probably means that it is not similar enough to contribute much. Either way the repulsion hypothesis seems to fit the most appropriately given the context of the rest of the experiments, as well as the ability for the midpoint case to exist with it. Whether we are incorrect in this notion, both are cases of NNP preserving distances in some form so the fix remains the same.

Understanding Diffusion

The results in the above sections answer: *How can we understand the diffusion or limited quality in projection?*. We can understand the diffusion as a combination of distance from the known and the distance preservation in the network's formulation.

This is in line with the causes we outlined from our exploration in the first few sections, where some points seemed to be far away while others just seemed to be problematic for no visible reason.

3.7.7 Improving Deep Learned Multidimensional Projections

The next question: *How can we improve the projections once we have this understanding?* is only partially fulfilled at this stage. We have understood some causes, and applied fixes to these causes. For 'distance from the known' as a cause we saw some improvements, specifically in visual clarity. We specify visual clarity instead of performance, because although we see some changes in the results of Sec. 3.6.2 these are not as prominently reflected in the quality metrics. Along with our need to aggregate the quality metrics to outline our error, there seems to be a case to be made that there is an easier or more straightforward metric which could help us identify the error, and evaluate our projections more accurately; perhaps some modification of the distance colormap. Regardless there seems to be some discrepancy between what we hope the existing quality metrics will measure and what they actually measure. Since the overall goal is to improve NNP to reach a level similar to t-SNE, we continue despite the small improvements in Sec. 3.6.2.

We found several insights pointing to various potential causes of our diffusion problems during the investigation of NNP, but we could not find a way to apply this insight to fundamentally improve NNP. So instead we try to overhaul NNP by introducing a new mechanism to the internal distance preservation function that we find causes error.

Chapter 4

Improving Deep Learning through Neighborhood Preservation

We have examined the architecture of NNP and have not found any specific area of focus, concluding that there must be some cause inherent to the structure of the projection with regard to its implementation. We hypothesize that this cause is to do with NNP wanting to preserve distances. To examine if this is the case, we try a new implementation of NNP, that seeks to mimic the strong visual separation of data clusters produced by t-SNE. t-SNE achieves this by essentially considering the preservation of *neighborhoods* rather than of point-pair distances. We apply this consideration to NNP in an attempt to harmonize the two methods and remove the problems we have seen within the architecture. We show Figure 4.1 to contrast the performance of these two methods with regards to the problems we have seen so far in this thesis.

The following chapter describes the implementation of NNP with neighbors as detailed in Modrakowski et al. [27]. [1]

4.1 Introducing KNNP

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 $(\mathbf{x}, P(\mathbf{x}))$ 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 [22] 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 [22].

We tune the hyperparameters of the KNNP model following the results in [3]. 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 Figure 2.6 aside from the input and output layers which are scaled so that each input layer containing K nD points outputs a single 2D point.

4.2 KNNP Evaluation

We next compare the KNNP method introduced in this chapter with the original NNP method using the optimized hyperparameter settings and with the ground-truth t-SNE projection. For this, we use the four quality metrics in Sec. 2.1.2. In addition to the MNIST dataset (Sec. 3.1), we add two more datasets to the comparison, namely:

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

[1] Work described in this chapter has been submitted for publication in the CCIS Series of books with Springer. [27]

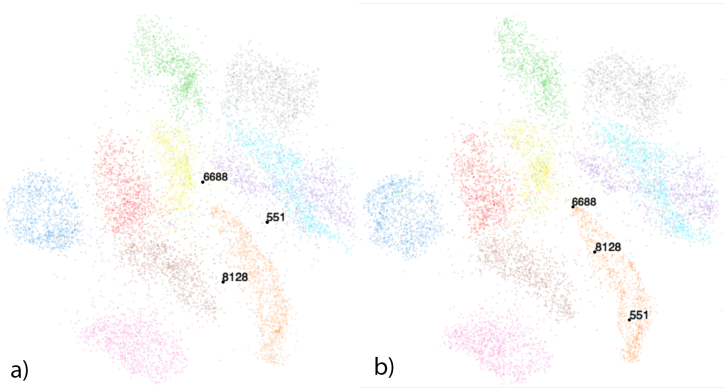


Figure 4.1: The placement of the points in (Figure 3.50-3.51) in NNP (a) and in KNNP (b)

IMDB Movie Review [29]: 25K movie reviews from which 700 features were extracted using TF-IDF [30], a standard method in text processing.

We next show the performance of KNNP *vs* NNP and t-SNE, for training data (Sec. 4.3) and test data (Sec. 4.4). We also show how quality depends on the training set size (Sec. 4.5) and evaluate KNNP's speed *vs* other techniques (Sec. 4.6). Finally, we show actual projection plots computed by KNNP, NNP, and t-SNE (Sec. 4.7). We present a subset of our results; the remainder is found in the supplemental material of [27].

4.3 Quality on training data

Figure 4.6 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.

4.4 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 4.7 shows that KNNP gets the largest quality boost *vs* NNP for $K = 5$ neighbors again. As in Fig. 4.6, 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 *explicitly* optimize a given quality metric.

4.5 Quality as function of training set size

Figure 4.2 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.

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

higher than NNP
same as NNP
lower than NNP

Figure 4.2: 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.

4.6 Computational scalability

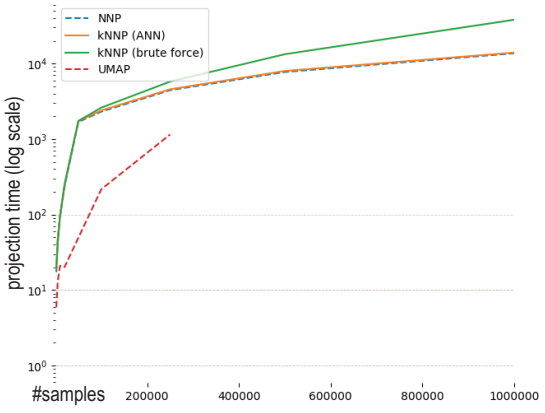


Figure 4.3: Comparison of training times between parametric techniques

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 4.4a 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 4.4b adds *non-parametric* techniques to the comparison, specifically MDS [7], t-SNE, LSP [6], and LAMP [8]. We see the same trend as before. Also, we see that KNNP is faster than all non-parametric techniques. Figure 4.3 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 the same speed, both are faster than KNNP with brute-force search.

4.7 Projection scatterplots

Figure 4.5 shows samples of scatterplots created with t-SNE, NNP, and KNNP with ANN and brute force search. We see that KNNP creates scatterplots which are less 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.

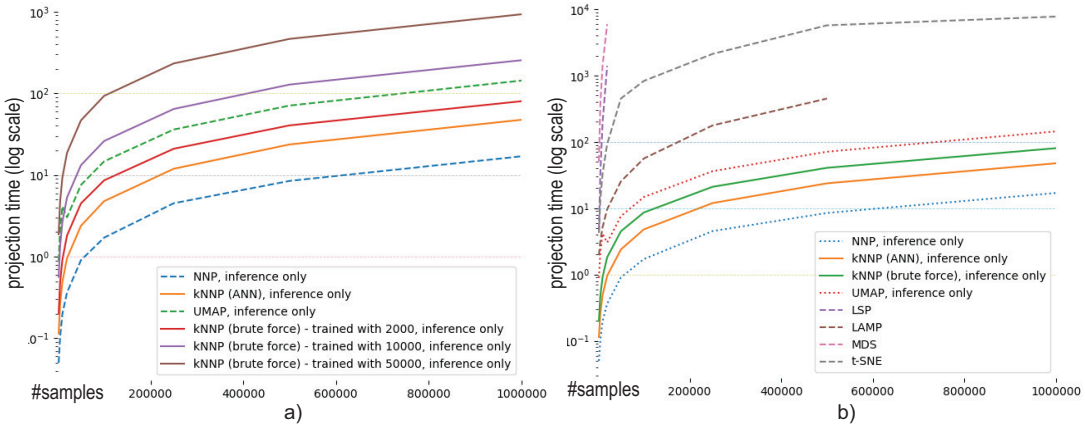


Figure 4.4: Projection times for parametric techniques only (a) and for parametric and non-parametric techniques (b)

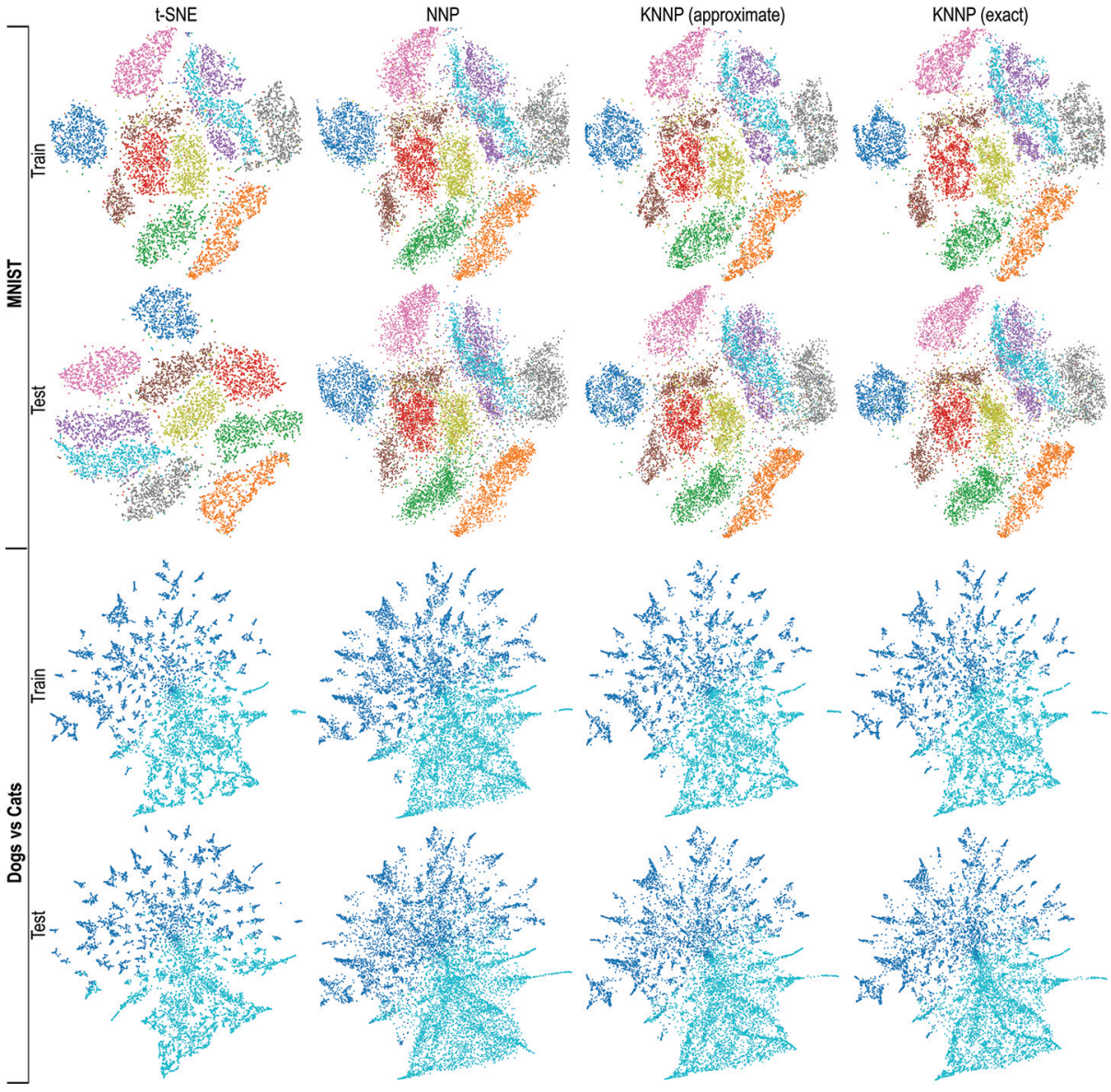


Figure 4.5: Projections created by t-SNE, KNN, and KNNP (approximate and exact search variants) on MNIST and Dogs *vs* Cats datasets during training and testing (inference).

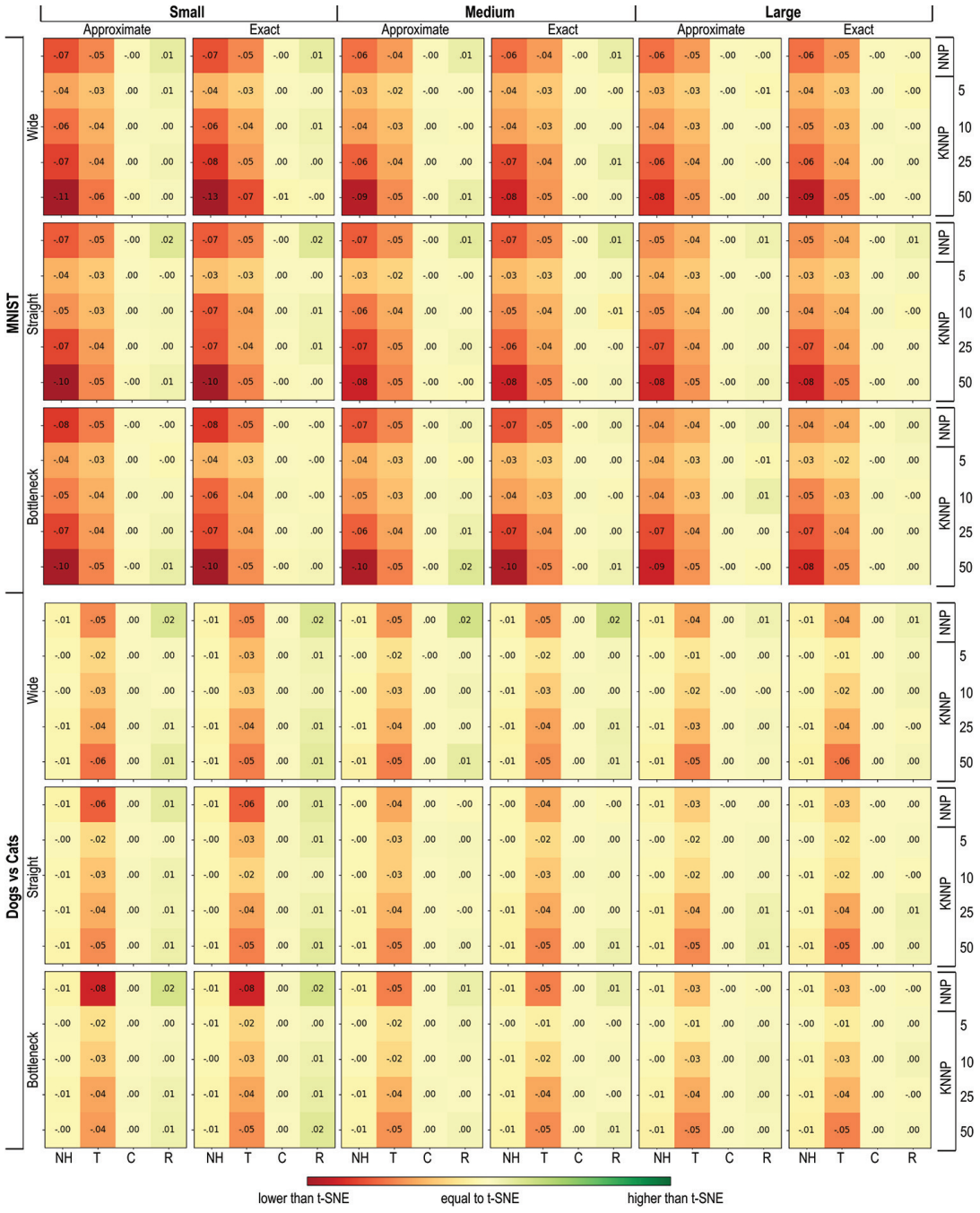


Figure 4.6: Comparison of the difference in four quality metrics NH, T, C, and R between t-SNE and NNP, respectively t-SNE and KNNP. The comparison is done on the MNIST and Dogs *vs* Cats datasets, for five K values, using both exact and approximate search, for three architecture styles (wide, straight, bottleneck), each having three sizes (small, medium, large). Red colors indicate cases which are farthest below t-SNE's quality.

Figure 4.7: Comparison of quality metrics of KNNP *vs* NNP for the same datasets, architectures, and parameters as in Fig. 4.6. Green indicates cases where KNNP performs better than NNP.

Chapter 5

Conclusion

5.1 Overview

We presented an in-depth summary of our experiments in this thesis, aimed at assessing issues and improving the quality of dimensionality reduction using deep learning. Specifically we aim to reduce the diffusion in the projections, as a means of answering our research question: How can we *improve the quality of deep learned multidimensional projections?*

We answer this question in two parts, based on sub-questions: *How can we understand the diffusion or limited quality in projection?* and *How can we improve the projections once we have this understanding?*

5.2 Part 1: Understanding Diffusion

We set out to understand the diffusion based on two fronts: the input and the architecture. These are the two components to the network. The input component could be broken down further into: the data and the training projection.

We isolated some problems in the input, determining where NNP worked well and where NNP worked poorly. Based on neighbors we were able to understand properties of what NNP did and did not learn. We gained some understanding of possible causes of diffusion, selecting 'Distance from the known' and 'Distance preservation' as the two we would pursue based on our understanding.

5.2.1 Is Distance from the Known the Reason for Diffusion?

We select 'distance from the known' as our focus cause, and examine it in more detail. Finding that the distance is strongly correlated with the error. We applied some fixes to this which involved modifying the input through ordering the training data and cleaning the training projection. We found some success in this, but we did not reach the goal of competing with t-SNE. In applying this fix we gained more understanding of this distance, and were able to conclude that distance is not the root cause of diffusion.

5.2.2 Which parts of the network are responsible for which parts of the projection?

We then select 'distance preservation' as our focus cause, and attempt to isolate problems in the architecture of the network, by examining the activations. We did not find significant evidence of discriminatory patches of neurons, but we did find that there was more dispersed activity in the points which are not in the center of clusters. We used this discovery to theorize that the activations are diluted because the network is explicitly trying to differentiate points as a product of its implementation. We further explored this implementation problem to choose the most likely from the two hypothesis we had so far. We did this by finding points which do not appear to be pulled away by any distantly placed point with similar learned features and concluded that this must be a result of NNP preserving the distances from the nearby points.

5.3 Part 2: How can we improve the projections once we have this understanding?

With all of our experiments complete, we have gained understanding. In some cases we iteratively explored causes in order to improve this understanding in those areas. We found that the 'Distance from the Known' is not the only cause, but also that it is entwined with the activation dilution.

We were unable to pinpoint a cause for these specific problems in the architecture, so we are unable to apply tweaks to layers or neurons that we wanted to perform. As a result we explore KNNP: an approach which tries to prioritize neighborhoods in much the same was as t-SNE does. With this we see some improvement quantitatively in the metrics, and a large increase in visual clarity qualitatively with regards to diffusion.

Figure 5.1 shows this visual improvement, particularly in the triangular base.

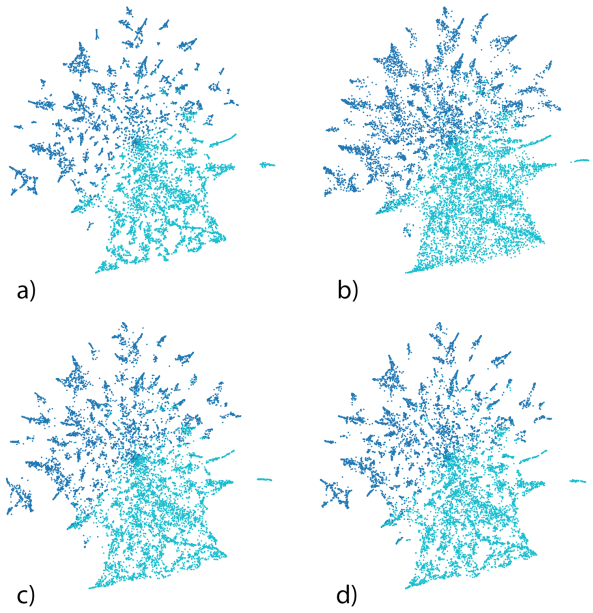


Figure 5.1: Projections created by t-SNE(a), NNP (b) and KNNP (exact search variant)(c) and KNNP (approximate search variant)(d) on Dogs *vs* Cats during training (Large/Wide).

Chapter 6

Future Work

In this chapter we detail the future work that has come into suggestion as a result of this thesis.

Quality Metrics
We examined three quality metrics, trustworthiness, continuity and neighborhood hit in the proposed solution, with the addition of Shepard's diagram correlation coefficient in the additional experiments. However, through our exposure to these metrics we find that they do not accurately provide a quantitative measure of the visual quality of the graph. For example some visible improvements seem to lead to minuscule quality improvements, so we are interested in the idea of applying a wider variety of quality metrics in the event that some may be more appropriate for outlining the exact properties we desire.

To fit closer to a qualitative evaluation of diffusion, we aggregated our metrics. This idea could be generalized further. We formed this aggregation as a simple combination of these metrics into one. By weighting these three metrics, we can more specifically tune the error to exclude some points that we would not like to see as error. One such case is the misclassifcations in the clusters which harshly reduce neighborhood hit values regardless of correct placement.

Distance and Loss Functions
We found in Sec.3.6.1 that we had correlation with the distances between the activations and the error. We theorize that incorporating this measure into the loss function or as some kind of error signal, may help the network learn to be aware of it. Similar to this, if we could bias the network, or manipulate the weights in order to force the network to bring points closer to the similar values rather than pushing points away, this may be helpful in bringing the diffusion closer to the clusters. The quality metrics, may be suitable as an error signal as well, more directly relating to the network what we want.

Iterative Pruning with Error Metrics
An extension of this idea is that if we are unable to incorporate the quality metrics as an error signal, we compute the quality metrics at each step of some stage of advanced training, and prune any weight modifications that would decrease the quality metrics. The idea being that if we only change weights that would improve the projection, we could end up with an optimal model with regard to quality metrics rather than accuracy to a training label.

KNNP
The current implementation of KNNP, as of writing this paper considers the neighbors which are most similar by an exact or approximate distance matrix. It may be the case that these neighborhoods could be specifically tailored in a more preferred way, such as tailoring each neighborhood to fit the point's desired location rather than it's most similar location. The most obvious way to do this is by manually tailoring these groups for training. If this process shows results, the process could be made implicit through means such as quality metrics, as we had done for removing points.

Furthermore, exploring KNNP with the same methods detailed here could be helpful in examining if the causes of diffusion have changed or if the causes we see here were simply obscuring another root cause.

Bibliography

[1] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. nov, pp. 2579–2605, 2008. Pagination: 27.

[2] M. Espadoto, N. S. T. Hirata, and A. C. Telea, "Deep learning multidimensional projections," 2019.

[3] M. Espadoto, N. Hirata, A. Falcão, and A. Telea, "Improving neural network-based multidimensional projections," pp. 29–41, 01 2020.

[4] M. Espadoto, R. M. Martins, A. Kerren, N. S. T. Hirata, and A. C. Telea, "Towards a quantitative survey of dimension reduction techniques.," *IEEE transactions on visualization and computer graphics*, 2019.

[5] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[6] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz, "Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 3, pp. 564–575, 2008.

[7] W. S. Torgerson, "Theory and methods of scaling. new york: John wiley and sons, inc., 1958. pp. 460," *Behavioral Science*, vol. 4, no. 3, pp. 245–247, 1959.

[8] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato, "Local affine multidimensional projection," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2563–2571, 2011.

[9] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[10] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018.

[11] E. Pekalska, D. de Ridder, R. P. W. Duin, and M. A. Kraaijveld, "A new method of generalizing sammon mapping with application to algorithm speed-up," 1999.

[12] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," *Distill*, 2016.

[13] T. Spinner, U. Schlegel, H. Schafer, and M. El-Assady, "explainer: A visual analytics framework for interactive and explainable machine learning," *IEEE Transactions on Visualization and Computer Graphics*, p. 1–1, 2019.

[14] P. Rauber, S. Fadel, A. Falcão, and A. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, pp. 1–1, 01 2016.

[15] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau, "Activis: Visual exploration of industry-scale deep neural network models," 2017.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 2016.

[17] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks," 2016.

[18] A. Harley, "An interactive node-link visualization of convolutional neural networks," pp. 867–877, 12 2015.

[19] E. Brown, J. Liu, C. Brodley, and R. Chang, "Dis-function: Learning distance functions interactively," pp. 83–92, 10 2012.

[20] F. L. Dennig, T. Polk, Z. Lin, T. Schreck, H. Pfister, and M. Behrisch, "Fdive: Learning relevance models using pattern-based similarity measures," *CoRR*, vol. abs/1907.12489, 2019.

[21] R. Martins, D. Coimbra, R. Minghim, and A. Telea, "Visual analysis of dimensionality reduction quality for parameterized projections," *Computers & Graphics*, vol. 41, 06 2014.

[22] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," 2016.

[23] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *AT&T Labs http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[24] J. Elson, J. J. Douceur, J. Howell, and J. Saul, "Asirra: a CAPTCHA that exploits interest-aligned manual image categorization," in *Proc. ACM CCS*, pp. 366–374, 2007.

[25] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *CoRR*, vol. abs/1505.06798, 2015.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR*, pp. 248–255, 2009.

[27] T. S. Modrakowski, M. Espadoto, A. X. Falcao, N. S. T. Hirata, and A. C. Telea, "Improving deep learning projections by neighborhood analysis," 2020.

[28] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[29] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "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.

[30] G. Salton and M. J. McGill, *Introduction to modern information retrieval*. McGraw-Hill, 1986.