Contrastive learning unifies $t$-SNE and UMAP

Sebastian Damrich  
IWR at Heidelberg University  
sebastian.damrich@iwr.uni-heidelberg.de

Jan Niklas Böhm  
University of Tübingen  
jan-niklas.boehm@uni-tuebingen.de

Fred A. Hamprecht  
IWR at Heidelberg University  
fred.hamprecht@iwr.uni-heidelberg.de

Dmitry Kobak  
University of Tübingen  
dmitry.kobak@uni-tuebingen.de

Abstract

Neighbor embedding methods $t$-SNE and UMAP are the de facto standard for visualizing high-dimensional datasets. They appear to use very different loss functions with different motivations, and the exact relationship between them has been unclear. Here we show that UMAP is effectively negative sampling applied to the $t$-SNE loss function. We explain the difference between negative sampling and noise-contrastive estimation (NCE), which has been used to optimize $t$-SNE under the name NCVis. We prove that, unlike NCE, negative sampling learns a scaled data distribution. When applied in the neighbor embedding setting, it yields more compact embeddings with increased attraction, explaining differences in appearance between UMAP and $t$-SNE. Further, we generalize the notion of negative sampling and obtain a spectrum of embeddings, encompassing visualizations similar to $t$-SNE, NCVis, and UMAP. Finally, we explore the connection between representation learning in the SimCLR setting and neighbor embeddings, and show that (i) $t$-SNE can be optimized using the InfoNCE loss and in a parametric setting; (ii) various contrastive losses with only few noise samples can yield competitive performance in the SimCLR setup.

1 Introduction

Low-dimensional visualization of high-dimensional data is a ubiquitous step in exploratory data analysis and the toolbox of visualization methods has been rapidly growing in the last years [1, 36, 38, 48, 49, 55]. Since all of these methods necessarily distort the true data layout [7, 21], it is beneficial to have different tools at one’s disposal. But only with a theoretic understanding of the aims of and relationships between different methods, practitioners can make informed decisions about which visualization to use for which purpose and how to interpret the results.

The state of the art for non-parametric, non-linear dimensionality reduction is based on the neighbor embedding framework [19]. The two most popular examples of neighbor embedding methods are $t$-SNE [53] and UMAP [36]. Prior work [5, 11, 26] has peeled away layers of complexity of $t$-SNE and UMAP to separate key ingredients from less important design choices. The main conceptual difference between $t$-SNE and UMAP lies in their loss functions, which are motivated very differently. Recent work [5, 11] found that UMAP’s application of negative sampling [37], avoiding the quadratic number of repulsive interactions between embedding points, drastically alters its loss function and decreases the repulsion compared to $t$-SNE. But a conceptual connection between their loss functions is still missing.

Our work fills this gap. We describe a new, precise connection between negative sampling and noise-contrastive estimation [14, 15], which has recently been applied to $t$-SNE under the name...
NCVis [2] (Sec. 4), resulting in a very efficient implementation. Our analysis gives rise to a spectrum of ‘contrastive’ neighbor embedding methods (Fig. 1) akin to that in [5], interpolating between UMAP and NCVis/t-SNE (Sec. 5). We demonstrate that UMAP can be seen as negative sampling applied to the t-SNE problem (Sec. 6) and explain why UMAP embeddings differ from t-SNE ones. Finally, we develop a unified PyTorch framework for contrastive parametric and non-parametric neighbor embedding methods, including a new method based on the InfoNCE loss [51], popular in self-supervised learning [3, 6, 9, 18, 29, 43, 50, 51, 57], and show the relevance of all these losses in the ‘contrastive learning’ [16] setting of SimCLR [9] (Sec. 7).

2 Related work

One of the most popular methods for data visualization is t-SNE [33, 53, 54]. Recently developed NCVis [2] employs noise-contrastive estimation [14, 15] to approximate t-SNE in a sampling-based way. UMAP [36], motivated by sophisticated mathematical theory, has matched t-SNE’s popularity at least in computational biology [4], and also uses sampling-based optimization, namely negative sampling [37]. UMAP sparked development of further related methods, e.g., TriMAP [1] optimizes its embedding by sampling triplets, while PacMAP [55] recommends using mid-range pairs of points.

Given their success, t-SNE and UMAP have been scrutinized to find out which aspects are essential to their performance. On the one hand, prior work [26, 32] found initialization to be important for both methods and strongly influencing the resulting global structure. On the other hand, the exact choice of the low-dimensional similarity kernel [5] or weights of the k-nearest-neighbor graph [11] are largely inconsequential. Both algorithms have similar relevant hyper-parameters such as, e.g., the heavy-tailedness of the similarity kernel [27, 58].

The central difference between t-SNE and UMAP is in their loss functions, which have been studied in [5, 11, 55], but never conceptually connected. We achieve this by deepening the link between negative sampling (NEG) and noise-contrastive estimation (NCE).
NEG was introduced as an ad hoc replacement for NCE in the context of learning word embeddings [37]. The relationship between NEG and NCE has been discussed before [12, 20, 31, 35, 45], but here we go further and provide the precise meaning of NEG: We show that, unlike NCE, NEG learns a model proportional but not equal to the true data distribution. This allows us to explain the qualitative difference between NEG and NCE, which we demonstrate using neighbor embeddings.

Both t-SNE and UMAP have parametric versions [46, 52] with very different implementations. Here we present a unified PyTorch framework for non-parametric and parametric contrastive neighbor embedding approaches. It encompasses UMAP, NEG, as well as t-SNE approximations both with NCE (like NCVis) and the InfoNCE loss [22, 47, 51], which has not been applied to neighbor embeddings. Moreover, we show that all of the mentioned loss functions (NCE/InfoNCE/NEG) can work similarly well in the SimCLR setting [9].

3 Background

3.1 Noise-contrastive estimation (NCE)

The goal of parametric density estimation is to fit a parametric model $q_{\theta}$ to iid samples $s_1, \ldots, s_N$ from an unknown data distribution $p$ over a space $X$. For maximum likelihood estimation (MLE) the parameters $\theta$ are chosen to maximize the log-likelihood of the observed samples

$$
\theta^* = \arg\max_\theta \sum_{i=1}^N \log(q_{\theta}(s_i)).
$$

(1)

This approach crucially requires $q_{\theta}$ to be a normalized model. It is otherwise trivial to increase the likelihood arbitrarily by scaling $q_{\theta}$. To circumvent the expensive computation of the partition function $Z(\theta) = \sum_{x \in X} q_{\theta}(x)$, Gutmann and Hyvärinen [14, 15] introduced NCE. It turns the unsupervised problem of density estimation into a supervised problem in which the data samples need to be identified from a set containing the $N$ data samples and $m$ times as many noise samples $t_1, \ldots, t_{mN}$ drawn from a noise distribution $\xi$, e.g., the uniform distribution over $X$. Briefly, NCE fits $\theta$ by maximizing the logarithm of the posterior probabilities $P(\text{data} \mid x) = q_{\theta}(x)/(q_{\theta}(x) + m\xi(x))$ and $P(\text{noise} \mid x) = 1 - P(\text{data} \mid x)$, or, equivalently, by minimizing the binary cross-entropy between the true class assignment and posterior probabilities (Appendix A.1):

$$
\theta^* = \arg\min_\theta \left[ -\sum_{i=1}^N \log \left( \frac{q_{\theta}(s_i)}{q_{\theta}(s_i) + m\xi(s_i)} \right) - m\sum_{i=1}^N \log \left( 1 - \frac{q_{\theta}(t_i)}{q_{\theta}(t_i) + m\xi(t_i)} \right) \right].
$$

(2)

The key advantage of NCE is that the model does not need to be explicitly normalized by the partition function, but nevertheless learns to equal the data distribution $p$ and hence be normalized:

**Theorem 1** ([14, 15]). Suppose there exists some $\theta^*$ such that $q_{\theta^*} = p$. Then $\theta^*$ is a minimum of

$$
\mathcal{L}^{\text{NCE}}(\theta) = -E_{x \sim p} \log \left( \frac{q_{\theta}(x)}{q_{\theta}(x) + m\xi(x)} \right) - mE_{t \sim \xi} \log \left( 1 - \frac{q_{\theta}(t)}{q_{\theta}(t) + m\xi(t)} \right)
$$

(3)

and any other minimum $\tilde{\theta}$ also satisfies $q_{\tilde{\theta}} = p$. In addition, if $\xi(x)$ is non-zero wherever $p(x)$ is non-zero, then these are the only extrema of $\mathcal{L}^{\text{NCE}}(\theta)$.

In NCE, the model typically includes an optimizable normalization parameter $Z$ which we emphasize by writing $q_{\theta, Z} = q_{\theta}/Z$. But importantly, Thm. 1 applies to any model $q_{\theta}$ that is able to match the data distribution $p$, even if it does not contain a learnable normalization parameter.

In the setting of learning language models, Jozefowicz et al. [22] proposed a different version of NCE, called InfoNCE. Instead of classifying samples as data or noise, the aim here is to predict the position of a data sample in an $(m+1)$-tuple containing $m$ noise samples and one data sample. In other words, InfoNCE replaces the binary classification of NCE with a $(m+1)$-class classification (Appendix A.2). For the uniform noise distribution $\xi$, this yields the expected loss function

$$
\mathcal{L}^{\text{InfoNCE}}(\theta) = -E_{x \sim p, x_1, \ldots, x_m \sim \xi} \log \left( \frac{q_{\theta}(x)}{q_{\theta}(x) + \sum_{i=1}^m q_{\theta}(x_i)} \right).
$$

(4)

Ma and Collins [35] showed that an analogue of Thm. 1 applies to InfoNCE.
3.2 Neighbor embeddings

Neighbor embeddings (NE) [19] are a group of dimensionality reduction methods, including UMAP [36], NCVis [2], and t-SNE [54], that aim to find a low-dimensional embedding $e_1, \ldots, e_n \in \mathbb{R}^d$ of high-dimensional input points $x_1, \ldots, x_n \in \mathbb{R}^D$, with $D \gg d$ and usually $d = 2$ for the purpose of visualization. NE methods define a notion of similarity over pairs of input data points which encodes the neighborhood structure and informs the low-dimensional embedding.

The exact high-dimensional similarity distribution differs between the NE algorithms, but recent work found empirically [5] and theoretically [11] that t-SNE and UMAP results stay practically the same when using the binary symmetric $k$-nearest-neighbor graph ($sk$NN) instead of t-SNE’s Gaussian or UMAP’s Laplacian similarities. An edge $ij$ is in $sk$NN if $x_i$ is among the $k$ nearest neighbors of $x_j$ or vice versa. The high-dimensional similarity function is then given by $p(ij) = \mathbb{1}(ij \in sk$NN$)/|sk$NN$|$, where $|sk$NN$|$ denotes the number of edges in the $sk$NN graph and $\mathbb{1}$ is the indicator function. NCVis uses the same similarities.

There are further differences in the choice of low-dimensional similarity between t-SNE and UMAP, but [5] shows that they are negligible. Therefore, here we use the Cauchy kernel $\phi(d_{ij}) = 1/(d_{ij}^2 + 1)$ for all NE methods to transform distances $d_{ij} = \|e_i - e_j\|$ in the embedding space into low-dimensional similarities. We abuse the notation slightly by also writing $\phi(ij) = \phi(d_{ij})$.

All NE methods that we are discussing in this work can be cast in the framework of parametric density estimation. Here, $p$ is the data distribution to be approximated with a model $q_\theta$, meaning that the space $X$ on which both $p$ and $q_\theta$ live is the set of all pairs $ij$ with $1 \leq i < j \leq n$. The embedding positions $e_1, \ldots, e_n$ become the learnable parameters $\theta$ of the model $q_\theta$. For t-SNE, NCVis, and UMAP, $q_\theta$ is proportional to $\phi(\|e_i - e_j\|)$, but the proportionality and the loss functions are different.

**t-SNE** uses MLE and therefore requires a normalized model $q_\theta(ij) = \phi(ij)/Z(\theta)$, where $Z(\theta) = \sum_{k \neq l} \phi(kl)$ is the partition function. The loss function

$$L^{t-SNE}(\theta) = -\mathbb{E}_{ij \sim p} \log (q_\theta(ij)) = -\sum_{i \neq j} (p(ij) \log (\phi(ij))) + \log \left(\sum_{k \neq l} \phi(kl)\right)$$ (5)

is the expected log-likelihood of the embedding positions $\theta$, making t-SNE an instance of MLE. Usually t-SNE’s loss function is introduced as the Kullback-Leibler divergence between $p$ and $q_\theta$, which is equivalent as the entropy of $p$ does not depend on $\theta$.

**NCVis** uses NCE and optimizes the expected loss function

$$L^{NCVis}(\theta, Z) = -\mathbb{E}_{ij \sim p} \log \left(\frac{q_\theta Z(ij)}{q_\theta Z(ij) + m \xi(ij)}\right) - m \mathbb{E}_{ij \sim \xi} \log \left(1 - \frac{q_\theta Z(ij)}{q_\theta Z(ij) + m \xi(ij)}\right),$$ (6)

where $q_\theta Z(ij) = \phi(ij)/Z$ with learnable $Z$ and $\xi$ is approximately uniform (see Appendix C).

According to Thm. 1, NCVis has the same optimum as t-SNE and can hence be seen as a sampling-based approximation of t-SNE. Indeed, we found that $Z$ in NCVis and the partition function $Z(\theta)$ in t-SNE converge approximately to the same value (Fig. S6).

**UMAP**’s expected loss function is derived in [11]:

$$L^{UMAP}(\theta) = -\mathbb{E}_{ij \sim p} \log (q_\theta(ij)) - m \mathbb{E}_{ij \sim \xi} \log (1 - q_\theta(ij)),$$ (7)

with $q_\theta(ij) = \phi(ij)$ and $\xi$ is approximately uniform, see Appendix C This is the effective loss function actually implemented in the UMAP algorithm, but note that it has only about $m/n$ of the repulsion compared to the loss stated in the original UMAP paper [36], as shown in [5] [11] (Appendix B).

In practice, the expectations in UMAP’s and NCVis’ loss functions are evaluated via sampling, like in Eq. (3). This leads to a fast, $O(n)$, stochastic gradient descent optimization scheme. Both loss functions are composed of an attractive term pulling similar data points (edges of the $sk$NN graph) closer together, and a repulsive term pushing random pairs of points further apart. Similarly, t-SNE’s loss yields attraction along the graph edges while repulsion arises through the normalization term.
4 From noise-contrastive estimation to negative sampling

In this section we work out the precise relationship between NCE and NEG, going beyond prior work [12][13][31][45]. NEG differs from NCE by its loss function and by the lack of the learnable normalization parameter $Z$. In our setting, NEG’s loss function amounts to

$$
\mathcal{L}_{\text{NEG}}(\theta) = -\mathbb{E}_{x \sim p} \log \left( \frac{q_\theta(x)}{q_\theta(x) + 1} \right) - m \mathbb{E}_{x \sim \xi} \log \left( 1 - \frac{q_\theta(x)}{q_\theta(x) + 1} \right). 
$$

(8)

In order to relate it to NCE’s loss function, we first generalize the latter in a way allowing to learn a model that is not equal but proportional to the true data distribution.

**Corollary 2.** Let $\tilde{Z}, m \in \mathbb{R}_+$. Assume that $\xi(x)$ is non-zero wherever $p(x)$ is non-zero and that there exist $\theta^*$ such that $q_{\theta^*} = \tilde{Z}p$. Then the generalized NCE loss function

$$
\mathcal{L}_{\text{NCE}}^{\tilde{Z}}(\theta) = -\mathbb{E}_{x \sim p} \log \left( \frac{q_\theta(x)}{q_\theta(x) + Zm\xi(x)} \right) - m \mathbb{E}_{x \sim \xi} \log \left( 1 - \frac{q_\theta(x)}{q_\theta(x) + Zm\xi(x)} \right) 
$$

(9)

has its only extrema where $q_\theta = Zp$.

**Proof.** The result follows from Thm. 1 applied to the model distribution $q_\theta := q_\theta / \tilde{Z}$.

It has been pointed out before [12][45] that for a uniform noise distribution $\xi(x) = 1/|X|$ and as many noise samples as the size of $X$ ($m = |X|$), the loss functions of NCE and NEG coincide, since $m\xi(x) = 1$. However, the main point of NCE and NEG is to use far fewer noise samples in order to attain a speed-up over MLE. Our Cor. 2 for the first time explains NEG’s behavior in this more realistic setting ($m \ll |X|$). If the noise distribution is uniform, the generalized NCE loss function with $\tilde{Z} = |X|/m$ equals the NEG loss function since $(|X|/m)m\xi(x) = 1$. By Cor. 2, any minimum $\theta^*$ of the NEG loss function yields $q_{\theta^*} = (|X|/m)p$, assuming that there are parameters that make this equation hold.

In other words, NEG aims to find a model $q_\theta$ that is proportional to the data distribution with the proportionality factor $|X|/m$ which is typically huge. This is different from NCE, which aims to learn a model equal to the data distribution. Therefore, the optimal parameters for NEG and NCE are typically different. While NCE uses a model $q_\theta/Z$ with learnable $Z$, we can interpret NEG as using a model $q_\theta/\tilde{Z}$ with fixed and very large normalization constant $\tilde{Z} = |X|/m$.

As a result, $q_\theta$ in NEG needs to attain much larger values to match the large $\tilde{Z}$. This can be illustrated in the setting of neighbor embeddings. Applying NEG to the neighbor embedding framework yields an algorithm that we call ‘Neg-t-SNE’. Recall that in this setting, the parameters $\theta = \{e_1, \ldots, e_n\}$ are the embedding positions and $|X| = {n \choose 2}$ is the number of pairs of points. Böhm et al. [5] found empirically that t-SNE’s partition function $Z(\theta)$ is typically between $50n$ and $100n$, while in NEG-t-SNE, $\tilde{Z} = \mathcal{O}(n^2)$ is much larger for modern big datasets. To attain the larger values of $\phi(ij)$ required by NEG, points that are connected in the $sk$NN graph have to move much closer together in the embedding than in t-SNE. Indeed, using our PyTorch implementation of Neg-t-SNE on the MNIST dataset, we confirmed that Neg-t-SNE (Fig. 1d) produced more compact clusters than t-SNE (Fig. 1c). See Appendix F and Alg. S1 for implementation details.

We emphasize that the conclusions of this section only hold because NEG does not contain a learnable normalization parameter $Z$. If it did, then such a learnable $Z$ would be able to absorb the term $\tilde{Z}$ in loss functions (9) and (10) while leaving the parameters $\theta$ unchanged.

5 Negative sampling spectrum

Varying the fixed normalization constant $\tilde{Z}$ in Eq. 9 has important practical effects that lead to a whole spectrum of embeddings in the NE setting. The original NEG loss function (8) corresponds to Eq. 9 with $\tilde{Z} = 1/m$. We still refer to the more general case of using an arbitrary $\tilde{Z}$ in Eq. 9 as ‘negative sampling’, and ‘Neg-t-SNE’ in the context of neighbor embeddings.

\footnote{We focus on the loss function, ignoring design choices of [37] specific for learning word_embeddings.}
Where NEG’s loss function (8) has fractions while holding \( \bar{Z} \) closely resembling the NCVis embedding (Fig. 1g). Similarly, setting \( \bar{Z} \) varies this parameter. We can explain this as follows. The repulsive term in the NCE loss (3) has a strength in the contrastive loss setting, while Böhm et al. [5] obtained the analogous spectrum by varying the \( m \) prefactor alters the effective loss. The UMAP paper [36] referred to their optimization as ‘negative sampling’, however UMAP’s loss function (7) does not look like the NEG loss (8), and hence it has been unclear if UMAP actually uses NEG in the sense of [37].

Nevertheless, we can rearrange the terms to make UMAP appear as a proper instance of NEG:

\[
q_\theta(ij) = \phi(ij) = \frac{1}{1 + d_{ij}^2} = \frac{1}{d_{ij}^2 + 1} = \frac{\hat{q}_\theta(ij)}{q_\theta(ij) + 1}.
\]  

(10)
Figure 3: NE embeddings of the MNIST dataset are qualitatively similar in the non-parametric (top row) and parametric (bottom row) settings. All panels use our PyTorch framework with $m = 5$ and batch size $b = 1024$.

where we introduced $\tilde{q}_\theta(ij) := 1/d_{ij}^2$. Put differently, UMAP uses NEG but not with a parametric model given by the Cauchy kernel, but instead with a parametric model $\tilde{q}_\theta$ which equals the squared inverse distance between embedding points.

For large embedding distances $d_{ij}$ both models behave similarly, but for nearby points they strongly differ: The inverse-square kernel $1/d_{ij}^2$ diverges when $d_{ij} \to 0$, whereas the Cauchy kernel $1/(1+d_{ij}^2)$ does not. Despite this qualitative difference, we found empirically that UMAP embeddings look very similar to Neg-t-SNE embeddings at $\bar{Z} = |X|/m$, see Figs. 1d and 1h for the MNIST example.

To explain this observation, it is instructive to compare the loss terms between Neg-t-SNE and UMAP: The attractive term amounts to $-\log(1/(d_{ij}^2 + 1)) = \log(1 + d_{ij}^2)$ for UMAP and $-\log[1/(d_{ij}^2 + 1)/(1/(d_{ij}^2 + 1) + 1)] = \log(2 + d_{ij}^2)$ for Neg-t-SNE, while the repulsive term equals $\log((1 + d_{ij}^2)/d_{ij}^2)$ and $\log((2 + d_{ij}^2)/(1 + d_{ij}^2))$, respectively. While the attractive terms are very similar, the repulsive term for UMAP diverges at zero but that of Neg-t-SNE does not (Fig S3).

This divergence introduces numerical instability into the optimization process of UMAP. In fact, UMAP employs several optimization tricks in order to overcome these instabilities. One of the tricks is annealing the learning rate down to zero, so that in the last optimization epochs the learning rate becomes very small.

We found that UMAP strongly depends on this annealing (Figs 2a, 2b). Without it, clusters appear fuzzy as noise pairs can experience very strong repulsion and get catapulted out of their cluster (Fig. 2a). While Neg-t-SNE also benefits from the annealing scheme (Fig. 2d), it produces a very similar embedding even without any annealing (Fig. 2c). Thus, UMAP’s effective choice of the $1/d_{ij}^2$ kernel makes it less numerically stable and more dependent on optimization tricks, compared to Neg-t-SNE. See Appendix D for more details.

Our conclusion is that at its heart, UMAP is NEG applied to the $t$-SNE framework. UMAP’s sampling-based optimization is much more than a mere optimization trick; it enables us to connect it theoretically to $t$-SNE. When UMAP’s loss function is seen as an instance of NEG, UMAP does not use the Cauchy kernel but rather the inverse-square kernel. However, this does not make a strong difference due to the learning rate decay. As discussed in Sec. 4, the fixed normalization constant $\bar{Z}$ in Neg-t-SNE/UMAP is much larger than the learnt $Z$ in NCVis or the partition function in $t$-SNE. This explains why UMAP pulls embedding points closer together than both NCVis and $t$-SNE and is the reason for the typically more compact clusters in UMAP embeddings [5].

[3] Shortly before the submission of this manuscript, a pull request to UMAP’s GitHub repository changed the effective kernel of parametric UMAP to the Cauchy kernel, in order to overcome numerical instabilities via an ad hoc fix, see Appendix D.
7 Contrastive neighbor embeddings and contrastive self-supervised learning

Contrastive self-supervised representation learning \([3, 6, 9, 18, 29, 43, 50, 51, 57]\) and ‘contrastive neighbor embeddings’ (a term we suggest for NCVis, Neg-t-SNE, UMAP, etc.) are conceptually very similar. The key difference is that the latter use a fixed \(k\)NN graph to find pairs of similar objects, while the former rely on data transformations or data proximity to generate pairs of similar objects on the fly. Other differences include the representation dimension (\(\sim 128\) vs. 2), the use of a neural network for parametric mapping, or the flavor of contrastive loss (InfoNCE \([9, 51]\) vs. NCE / NEG). However, these other differences are not crucial, as we demonstrate in this section by using our unified PyTorch framework.

As an example, we demonstrate that \(t\)-SNE can also be optimized using the InfoNCE loss, resulting in InfoNC-\(t\)-SNE (Appendix \(F\) and Alg. \(S1\)). The result of InfoNC-\(t\)-SNE on MNIST (Fig. \(3g\)) was similar to the result of NCVis (Fig. \(3f\)). For the default number of noise samples, \(m = 5\), both algorithms produced embeddings that were visibly different from \(t\)-SNE proper (Fig. \(1f\)). Recent work employing the InfoNCE loss for self-supervised learning \([9]\) generally reports improved performance for more noise samples, in agreement with the theoretical results \([14, 15, 35]\). We observed that both NCVis and InfoNC-\(t\)-SNE visualizations using \(m = 500\) approximated \(t\)-SNE much better (Figs. \(S9k\) and \(S10k\)). Like \(t\)-SNE, but unlike the \(m = 5\) setting, \(m = 500\) required early exaggeration to prevent cluster fragmentation, see Figs. \(S9\) and \(S10\).

Next, we used our PyTorch framework to obtain parametric versions of all contrastive NE algorithms discussed here (NCVis, InfoNC-\(t\)-SNE, Neg-\(t\)-SNE, UMAP). We used a fully connected neural network with four linear layers and ReLU activations as a parametric \(\mathbb{R}^D \rightarrow \mathbb{R}^d\) mapping and optimized its parameters using Adam \([24]\) (Appendix \(F\)). We used batch size \(b = 1024\) and sampled all \(m\) negative samples from within the batch; the data set was shuffled each epoch before batching. Using all four loss functions, we were able to get parametric embeddings of MNIST that were qualitatively similar to their non-parametric versions (Fig. \(3\)). The parametric versions of NCE and InfoNCE produced much larger embeddings than their non-parametric counterparts, however the final loss values were very similar (Figs. \(S5a\) \(\leq S5b\)). For NCE, the larger scale of the parametric embedding was compensated by a smaller learned normalization parameter \(Z\), so that both parametric and non-parametric versions were approximately normalized (Fig. \(S5c\)). Our parametric UMAP implementation is very similar to the implementation of Sainburg et al. \([46]\). But our parametric, approximate \(t\)-SNE implementations are very different from the parametric \(t\)-SNE of Van Der Maaten \([52]\), which constructed separate \(k\)NN graphs within each batch and optimized the vanilla \(t\)-SNE loss function, whereas we use the full \(k\)NN graph and rely on NCE/InfoNCE losses.

Finally, as a proof-of-concept, we demonstrate that the different contrastive loss functions can all work for image-based self-supervised learning, and specifically NCE and NEG can work similarly well to InfoNCE in SimCLR \([9]\), using only \(m = 16\) noise samples. We used a SimCLR setup to train representations of the CIFAR-10 dataset \([28]\) using a ResNet18 \([17]\) backbone and a fully-connected projector head to 128 embedding dimensions. We used the same data augmentations to create pairs of similar images as in \([9]\), also constrained the output to the sphere \(S^{128}\), and used the exponential of the cosine similarity (Appendix \(F\)). NCE and NEG produced embeddings of similar quality as InfoNCE, measured by the classification accuracy on the ResNet output (Tab. \(1\)). In contrast to claims in the literature \([9]\), we could achieve competitive results with only a few, non-curated \([44, 56]\) noise samples. Future work is needed to perform more systematic benchmarks and to study the effects of the contrastive loss function on the SimCLR performance. Here we present these results only as a proof of principle.

8 Discussion and Conclusion

In this work, we studied the relationship between two popular unsupervised learning methods, noise-contrastive estimation (NCE) and negative sampling (NEG). We focused on their application to neighbor embeddings (NE) because this is an active and important application area, but also because NEs allow to directly visualize the NCE/NEG outcome and to form intuitive understanding of how different algorithm choices affect the result. Our study makes three conceptual advances.

First, we showed that NEG replaces NCE’s learnable normalization parameter \(Z\) by a large constant \(\bar{Z}\), forcing NEG to learn a scaled data distribution. In the NE setting, this led to the method Neg-\(t\)-SNE,
An important caveat is that Thm. 1 and Cor. 2 both assume that the model is rich enough to perfectly represent the data. More specifically, we argued that contrastive NEs are closely related to the contrastive self-supervised learning methods such as SimCLR [9], which can be seen as parametric InfoNC-t-SNE for learning representations in \( S^d \) based on the unobservable similarity graph implicitly constructed via data augmentations. We feel that this connection has been underappreciated, with the literature on NEs and on self-supervised contrastive learning staying mostly disconnected. To bridge the gap between these two worlds, we presented InfoNCE (16), NCE (16), and NEG (16) for different number of noise samples \( m \) (in parentheses) and with batch size \( b = 1024 \). Classification accuracy was computed on a test set, using the ResNet18 output \( H \in \mathbb{R}^{512} \). We report the mean and the standard deviation across three random seeds. In the first row, the classifier was trained using data augmentations [9].

|            | InfoNCE (2b – 2) | InfoNCE (16) | NCE (16) | NEG (16) |
|------------|------------------|--------------|----------|----------|
| Linear classifier (augm.) | 89.3 ± 0.4% | 91.6 ± 0.2% | 88.6 ± 0.3% | 89.6 ± 0.6% |
| Linear classifier | 89.3 ± 0.5% | 90.1 ± 0.6% | 87.6 ± 0.4% | 88.9 ± 0.5% |
| kNN classifier | 89.3 ± 0.4% | 89.5 ± 0.4% | 86.0 ± 0.1% | 86.7 ± 0.9% |

Table 1: Top-1 accuracies on CIFAR-10 representations learned with various contrastive learning losses for different number of noise samples \( m \) (in parentheses) and with batch size \( b = 1024 \). Classification accuracy was computed on a test set, using the ResNet18 output \( H \in \mathbb{R}^{512} \). We report the mean and the standard deviation across three random seeds. In the first row, the classifier was trained using data augmentations [9].
Acknowledgments and Disclosure of Funding

We thank Philipp Berens for comments and support as well as James Melville for discussions.

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, Germany’s Research Foundation) via SFB 1129 (240245660; S.D., F.A.H.) as well as via Excellence Clusters 2181/1 “STRUCTURES” (390900948; S.D., F.A.H.) and 2064 “Machine Learning: New Perspectives for Science” (390727645; J.N.B., D.K.), by the German Ministry of Education and Research (Tübingen AI Center, 01IS18039A; J.N.B., D.K.), and by the Cyber Valley Research Fund (D.30.28739; J.N.B.).

References

[1] E. Amid and M. K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets. arXiv preprint arXiv:1910.00204, 2019.

[2] A. Artemenkov and M. Panov. NCVis: Noise Contrastive Approach for Scalable Visualization. In Proceedings of The Web Conference 2020, pages 2941–2947, 2020.

[3] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. Advances in Neural Information Processing Systems, 33:12449–12460, 2020.

[4] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell. Dimensionality reduction for visualizing single-cell data using UMAP. Nature Biotechnology, 37(1):38–44, 2019.

[5] J. N. Böhm, P. Berens, and D. Kobak. Attraction-Repulsion Spectrum in Neighbor Embeddings. Journal of Machine Learning Research, 23(95), 2022.

[6] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. Advances in Neural Information Processing Systems, 33:9912–9924, 2020.

[7] T. Chari, J. Banerjee, and L. Pachter. The Specious Art of Single-Cell Genomics. bioRxiv, 2021.

[8] B. Charlier, J. Feydy, J. A. Glaunès, F.-D. Collin, and G. Durif. Kernel Operations on the GPU, with Autodiff, without Memory Overflows. Journal of Machine Learning Research, 22(74):1–6, 2021. URL http://jmlr.org/papers/v22/20-275.html.

[9] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In International Conference on Machine Learning, pages 1597–1607. PMLR, 2020.

[10] A. Coenen and A. Pearce. A deeper dive into UMAP theory. https://pair-code.github.io/understanding-umap/supplement.html, 2022. Accessed: 2022-05-11.

[11] S. Damrich and F. A. Hamprecht. On UMAP’s True Loss Function. Advances in Neural Information Processing Systems, 34, 2021.

[12] C. Dyer. Notes on Noise Contrastive Estimation and Negative Sampling. arXiv preprint arXiv:1410.8251, 2014.

[13] Y. Goldberg and O. Levy. word2vec Explained: Deriving Mikolov et al.’s Negative-Sampling Word-Embedding Method. arXiv preprint arXiv:1402.3722, 2014.

[14] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[15] M. U. Gutmann and A. Hyvärinen. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. Journal of Machine Learning Research, 13(2), 2012.
[16] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, 2:1735–1742, 2006.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[18] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[19] G. E. Hinton and S. Roweis. Stochastic Neighbor Embedding. *Advances in Neural Information Processing Systems*, 15, 2002.

[20] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

[21] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:28, 1984.

[22] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the Limits of Language Modeling. arXiv preprint arXiv:1602.02410, 2016.

[23] S. Kanton, M. J. Boyle, Z. He, M. Santel, A. Weigert, F. Sanchís-Calleja, P. Guijarro, L. Sidow, J. S. Fleck, D. Han, et al. Organoid single-cell genomic atlas uncovers human-specific features of brain development. *Nature*, 574(7778):418–422, 2019.

[24] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[25] D. Kobak and P. Berens. The art of using t-SNE for single-cell transcriptomics. *Nature Communications*, 10(1):1–14, 2019.

[26] D. Kobak and G. C. Linderman. Initialization is critical for preserving global data structure in both t-SNE and UMAP. *Nature Biotechnology*, pages 1–2, 2021.

[27] D. Kobak, G. Linderman, S. Steinerberger, Y. Kluger, and P. Berens. Heavy-Tailed Kernels Reveal a Finer Cluster Structure in t-SNE Visualisations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 124–139. Springer, 2019.

[28] A. Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, University of Toronto, 2009.

[29] P. H. Le-Khac, G. Healy, and A. F. Smeaton. Contrastive Representation Learning: A Framework and Review. *IEEE Access*, 8:193907–193934, 2020.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] O. Levy and Y. Goldberg. Neural Word Embedding as Implicit Matrix Factorization. *Advances in Neural Information Processing Systems*, 27, 2014.

[32] G. C. Linderman and S. Steinerberger. Clustering with t-SNE, Provably. *SIAM Journal on Mathematics of Data Science*, 1(2):313–332, 2019.

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

[34] I. Loshchilov and F. Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *International Conference on Learning Representations*, 2017.
[51] A. Van den Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. arXiv e-prints, pages arXiv–1807, 2018.

[52] L. Van Der Maaten. Learning a Parametric Embedding by Preserving Local Structure. In Artificial Intelligence and Statistics, pages 384–391. PMLR, 2009.

[53] L. Van Der Maaten. Accelerating t-SNE using Tree-Based Algorithms. Journal of Machine Learning Research, 15(1):3221–3245, 2014.

[54] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. Journal of Machine Learning Research, 9(11), 2008.

[55] Y. Wang, H. Huang, C. Rudin, and Y. Shaposhnik. Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMAP, and PaCMAP for Data Visualization. Journal of Machine Learning Research, 22:1–73, 2021.

[56] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. Sampling Matters in Deep Embedding Learning. In Proceedings of the IEEE International Conference on Computer Vision, pages 2840–2848, 2017.

[57] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised Feature Learning via Non-Parametric Instance Discrimination. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3733–3742, 2018.

[58] Z. Yang, I. King, Z. Xu, and E. Oja. Heavy-Tailed Symmetric Stochastic Neighbor Embedding. In Advances in Neural Information Processing Systems, pages 2169–2177, 2009.
A Probabilistic frameworks of NCE and InfoNCE

In this section we recap the probabilistic frameworks of NCE [14,15] and InfoNCE [22,51].

A.1 NCE

In NCE the unsupervised problem of parametric density estimation is turned into a supervised problem in which the data samples need to be identified in a set \( S \) containing \( N \) data samples \( s_1, \ldots, s_N \) and \( m \) times as many noise samples \( t_1, \ldots, t_{mN} \) drawn from a noise distribution \( \xi \), which can be (but does not have to be) the uniform distribution. In other words, we are interested in the posterior probability \( P(y|x) \) of element \( x \in S \) coming from the data (\( y = \text{data} \)) rather than from the noise distribution (\( y = \text{noise} \)).

The probability of sampling \( x \) from noise, \( P(x|\text{noise}) \), is just the noise distribution \( \xi \), and similarly \( P(x|\text{data}) \) is the data distribution \( p \). Since the latter is unknown, it is replaced by the model \( q_\theta \). Since \( S \) contains \( m \) times as many noise samples as data samples, the prior class probabilities are \( P(\text{data}) = 1/(m+1) \) and \( P(\text{noise}) = m/(m+1) \). Thus, the unconditional probability of an element of \( S \) is \( P(x) = (q_\theta(x) + m\xi(x))/(m+1) \). The posterior probability for classifying some given element \( x \) of \( S \) as data rather than noise is thus

\[
P(\text{data}|x) = \frac{P(x|\text{data})P(\text{data})}{P(x)} = \frac{q_\theta(x)}{q_\theta(x) + m\xi(x)}. \tag{11}\]

NCE optimizes the parameters \( \theta \) by maximizing the log-likelihood of the posterior class distributions, or, equivalently, by minimizing the negative log-likelihoods. This is the same as a sum over binary cross-entropy losses (Eq. 2 in the main text):

\[
\theta^* = \arg\min_\theta \left[ -\sum_{i=1}^N \log \left( \frac{q_\theta(s_i)}{q_\theta(s_i) + m\xi(s_i)} \right) - \sum_{i=1}^{mN} \log \left( 1 - \frac{q_\theta(t_i)}{q_\theta(t_i) + m\xi(t_i)} \right) \right]. \tag{12}\]

In expectation, we have the loss function (Eq. 3 in the main text)

\[
\mathcal{L}^{\text{NCE}}(\theta) = -\mathbb{E}_{s \sim p} \log \left( \frac{q_\theta(s)}{q_\theta(s) + m\xi(s)} \right) - m\mathbb{E}_{t \sim \xi} \log \left( 1 - \frac{q_\theta(t)}{q_\theta(t) + m\xi(t)} \right). \tag{13}\]

Since \( q_\theta(x)/(q_\theta(x) + m\xi(x)) = 1/[1 + \left(q_\theta(x)/(m\xi(x))\right)^{-1}] \), NCE’s loss function can also be seen as binary logistic regression loss function with \( \log \left( \frac{q_\theta(x)}{m\xi(x)} \right) \) as the input to the logistic function:

\[
\mathcal{L}^{\text{NCE}}(\theta) = -\mathbb{E}_{s \sim p} \log \left( \sigma \left( \log \left( \frac{q_\theta(s)}{m\xi(s)} \right) \right) \right) - m\mathbb{E}_{t \sim \xi} \log \left( 1 - \sigma \left( \log \left( \frac{q_\theta(t)}{m\xi(t)} \right) \right) \right), \tag{14}\]

where \( \sigma(x) = 1/(1 + \exp(-x)) \) is the logistic function.

A.2 InfoNCE

Again, we are casting the unsupervised problem of density estimation as a supervised classification problem. We consider a tuple of \( m + 1 \) samples \( T = (x_0, \ldots, x_m) \) one of which comes from the data and the rest from the noise distribution. Instead of classifying each sample independently as noise or data (as in Sec. [A.1]), here we are interested in identifying the position of the single data sample. This allows us to see the problem as a multi-class classification problem with \( m + 1 \) classes.

Let \( Y \) be the random variable that holds the index of the data sample. A priori, we have \( P(Y = k) = 1/(m + 1) \) for all \( k = 0, \ldots, m \). Moreover, conditioned on sample \( k \) coming from the data distribution, all other samples must come from the noise distribution, i.e., we have \( P(x_i|Y = k) = \xi(x_i) \) for \( i \neq k \). As the data distribution is unknown, we model it with \( P(x_k|Y = k) = q_\theta(x_k) \) as in Sec. [A.1]. This yields the likelihood of tuple \( T \) given the data index \( Y = k \)

\[
P(T|Y = k) = q_\theta(x_k) \prod_{i \neq k} \xi(x_i) = \frac{q_\theta(x_k)}{\xi(x_k)} \prod_{i=0}^m \xi(x_i). \tag{15}\]
Marginalizing over \( Y \), we obtain

\[
P(T) = \frac{1}{m+1} \prod_{i=0}^{m} \xi(x_i) \sum_{k=0}^{m} q_{\theta}(x_k) \xi(x_k)
\]  

(16)

Finally, we can compute the posterior via Bayes’ rule as

\[
P(Y = k|T) = \frac{P(T|Y = k)P(Y = k)}{P(T)} = \frac{q_{\theta}(x_k)}{\sum_{i=0}^{m} q_{\theta}(x_i)} \xi(x_k)
\]

(17)

where the last equality only holds for the uniform noise distribution. The InfoNCE loss is the cross-entropy loss with respect to the true position of the data sample, i.e., in expectation and for uniform \( \xi \) it reads:

\[
\mathcal{L}_{\text{InfoNCE}}(\theta) = - \mathbb{E}_{x \sim p, x_1, \ldots, x_m \sim \xi} \log \left( \frac{q_{\theta}(x)}{q_{\theta}(x) + \sum_{i=1}^{m} q_{\theta}(x_i)} \right).
\]

(18)

Similar to how the NCE loss can be seen as the binary logistic regression loss function (Sec. A.1), the InfoNCE loss can be viewed as the multinomial logistic regression loss function with the terms \( \log \left( \frac{q_{\theta}(x)}{\xi(x)} \right) \) entering the softmax function.

**B UMAP’s loss function**

In the original UMAP paper, McInnes et al. [36] define weights \( \mu(ij) \in [0,1] \) on the \( k \)NN graph and state that these shall be reproduced by the low-dimensional similarities \( \phi(ij) \) by means of a sum of binary cross-entropy loss functions, one for each edge \( ij \)

\[
- \sum_{ij} \left[ \mu(ij) \log(\phi(ij)) + (1 - \mu(ij)) \log(1 - \phi(ij)) \right].
\]

(19)

Indeed, this loss has its minimum at \( \mu(ij) = \phi(ij) \) for all \( ij \). However, it is of course not possible to achieve zero loss for any real-world data using the Cauchy kernel in two dimensions. Experiments show that using this loss function in practice leads to an excess of repulsion and consequently to very poor embeddings [5]. The actual UMAP implementation has much less repulsion due to the sampling of repulsive edges, see below.

As the weights \( \mu(ij) \) are only supported on the sparse \( k \)NN graph, most of the \( 1 - \mu(ij) \) terms are equal to one. To simplify the loss function, the UMAP paper replaces all \( 1 - \mu(ij) \) terms by 1, leading to the loss function

\[
- \sum_{ij} \left[ \mu(ij) \log(\phi(ij)) + \log(1 - \phi(ij)) \right].
\]

(20)

In the implementation, UMAP samples the repulsive edges, which drastically changes the loss [5] to the effective loss [11]

\[
- \sum_{ij} \left[ \mu(ij) \log(\phi(ij)) + \frac{m(d_i + d_j)}{2n} \log(1 - \phi(ij)) \right],
\]

(21)

where \( d_i = \sum_{j} \mu(ij) \) denotes the degree of node \( i \) and the number of negative samples \( m \) is a hyperparameter. By default, \( m = 5 \). Since \( d_i \approx \log(k) \), the effective loss only has about \( m \log(k)/n \) of the repulsion in the originally stated loss function. As a result, the \( \mu(ij) \) are not reproduced in the embedding space by this loss function.

We rewrite this effective loss function further to fit into our framework. The attractive prefactors \( \mu(ij) \) sum to \( \sum_{ij} \mu(ij) \), while the repulsive prefactors add up to \( m \) times this factor. Dividing the entire loss function by this term does not change its properties. But then, we can write the prefactors as probability distributions \( p(ij) = \mu(ij)/\sum_{ij} \mu(ij) \) and \( \xi(ij) = (p(i) + p(j))/n \) using \( p(i) = \sum_{j} p(ij) \). With this, we can write the effective loss function as

\[
- \sum_{ij} p(ij) \log(\phi(ij)) - m \sum_{ij} \xi(ij) \log(1 - \phi(ij)),
\]

(22)
or in the expectation form as

\[-E_{ij \sim p} \log (\phi(ij)) - mE_{ij \sim \xi} \log (1 - \phi(ij)),\]  

(23)

like we do in Eq. (7).

C Noise distributions

Here we discuss the various noise distributions used by UMAP, NCVis, and our framework. The main claim is that all these noise distributions are sufficiently close to uniform, even though their exact shape depends on the implementation details.

Since our actual implementation, as well as the reference implementations of UMAP and NCVis, considers edges \(ij\) and \(ji\) separately, we will do so from now on. Hence, there is now a total of \(E := 2|skNN|\) edges. We always assume that \(p(ij) = p(ji)\) and adding up the probabilities for both directions yields one: \(\sum_{i,j=1}^{n} p(ij) = 1\). For a given data distribution over pairs of points \(ij\), we define \(p(i) = \sum_{j=1}^{n} p(ij)\) so that \(\sum_{i} p(i) = 1\). As discussed in Appendix B of [11], the \(p(i)\) values are approximately constant when \(p(ij)\) is uniform on the skNN graph or proportional to the UMAP similarities.

UMAP’s noise distribution is derived in [11] and reads in our notation (Appendix B)

\[\xi(ij) = \frac{p(i) + p(j)}{2n}.\]  

(24)

Note that UMAP uses a weighted version of the skNN graph. Still, \(\xi\) is close to uniform, see Appendix B of [11].

The noise distribution of NCVis is also close to being uniform and equals [2]:

\[\xi(ij) = \frac{p(i)}{n}.\]  

(25)

This is a slightly different noise distribution than in UMAP and in particular it is asymmetric. However, we argue that in practice it is equivalent to the same noise distribution as in UMAP. The noise distribution is used in two ways in NCVis: for sampling noise samples and in the posterior class probabilities

\[P(data|ij) = \frac{q_{\theta, z}(ij)}{q_{\theta, z}(ij) + m\xi(ij)}.\]  

(26)

Both in the reference NCVis implementation and in ours, for the second role, the noise distribution is explicitly approximated by the uniform one and we use the posterior probabilities

\[P(data|ij) = \frac{q_{\theta, z}(ij)}{q_{\theta, z}(ij) + m\frac{1}{2|skNN|}}.\]  

(27)

Together with the symmetry of the Cauchy kernel this implies that the repulsion on the embedding vectors \(e_i\) and \(e_j\) from noise sample \(ij\) and \(ji\) is the same. As a result, the expectation

\[E_{ij \sim \xi} \log \left( 1 - \frac{q_{\theta, z}(ij)}{q_{\theta, z}(ij) + m\frac{1}{2|skNN|}} \right)\]  

(28)

is the same for \(\xi(ij) = p(i)/n\) and for UMAP’s noise distribution

\[\xi(ij) = \frac{p(i) + p(j)}{2n}.\]  

(29)

In our framework, the noise distribution is influenced by the batched training procedure (Alg. S1) because the negative samples can come only from the current training batch.

In every epoch, we first shuffle the set of directed edges of the skNN graph and then chunk it into batches. To emphasize, the batches consist of directed edges and not of the original data points. For each edge in a batch, we take its head and sample \(m\) indices from the heads and tails of all edges in the batch (excluding the already selected head) and use them as tails to form negative sample pairs.
To obtain a negative sample pair $ij$, the batch must contain some directed edge $ik$, providing the head of the negative sample, and some pair $lj$ or $jl$, providing the tail. We want to derive the expected number of times that a directed edge $ij$ is considered as a negative sample in a batch. For simplicity, let us assume that the number of batches divides the number of directed edges $E$. As the set of $sk$NN edges is shuffled every epoch, the expected number of pairs $ij$ as negative samples is the same for all batches.

Let us consider a batch $B$ of size $b$. We denote by $Y_{rs}$ the random variable that holds the number of times edge $rs$ appears in $B$. We also introduce random variables $Y_{r-s} = \sum_{i \neq s} Y_{is}$ and $Y_{r+s} = \sum_{i \neq r} Y_{ri}$. Let $p(r-s) = p(\neg sr) := p(r) - p(rs)$. For each occurrence of an $i$ as head of an edge in $B$, we sample $m$ tails to create negative samples uniformly from all head and tails in $B$ with replacement, but we prevent sampling the identical head $i$ as negative sample tail. If, however, the same node $i$ is part of the other edges in the batch, then it may be sampled and would create a futile negative sample $ii$. There are $m$ chances for creating a negative sample edge $ij$ for every head $i$ and any occurrence of $j$ in the batch. The number of heads $i$ in the batch is $Y_{ij} + Y_{i\neg j}$ and the number of occurrences of $j$ is $Y_{ij} + Y_{ji} + Y_{i\neg j} + Y_{j\neg i}$. Since we sample the tail of a negative sample pair uniformly with replacement, any of the occurrences of $j$ has probability $1/(2b - 1)$ to be selected. Hence, the expected number $N_{ij}$ of times that the ordered pair $ij$ with $i \neq j$ is considered as a negative sample in batch $B$ is

$$N_{ij} = m(Y_{ij} + Y_{i\neg j}) \frac{Y_{ij} + Y_{ji} + Y_{i\neg j} + Y_{j\neg i}}{2b - 1},$$  

(30)

Since a head $i$ may not choose itself to form a negative sample, the expected number of times that $ii$ appears as negative sample in the batch is

$$N_{ii} = m \sum_j Y_{ij} \frac{Y_{ij} - 1 + Y_{i\neg j} + Y_{ji} + Y_{j\neg ji}}{2b - 1}.$$  

(31)

Since the batches are sampled without replacement, the random variables $Y_{rs}$ are distributed according to a multivariate hypergeometric distribution, meaning that

$$\mathbb{E}(Y_{rs}) = bp(rs)$$  

(32)

$$\mathbb{E}(Y_{r-s}) = bp(\neg rs)$$  

(33)

$$\text{Var}(Y_{rs}) = b \frac{E - b}{E - 1} p(rs) (1 - p(rs))$$  

(34)

$$\text{Cov}(Y_{rs}, Y_{r-s}) = -b \frac{E - b}{E - 1} p(rs) p(\neg sr).$$  

(35)

We use these expressions and analogous ones together with the symmetries $p(rs) = p(sr)$ to compute (leaving out intermediate algebra steps) the expectation of $N_{ij}$ over the shuffles:

$$\mathbb{E}(N_{ij}) = \frac{mb}{2b - 1} \left( \frac{E - b}{E - 1} p(ij) + 2 \left( \frac{b - E - b}{E - 1} \right) p(i)p(j) \right).$$  

(36)

Since we sample $m$ negative samples for each positive sample and since each batch contains $b$ positive samples, we need to divide $\mathbb{E}(N_{ij})$ by $mb$ to obtain $\xi(ij)$:

$$\xi(ij) = \frac{1}{2b - 1} \left( \frac{E - b}{E - 1} p(ij) + 2 \left( \frac{b - E - b}{E - 1} \right) p(i)p(j) \right).$$  

(37)

Similarly,

$$\mathbb{E}(N_{ii}) = \frac{mb}{2b - 1} \left( - \frac{b - 1}{E - 1} p(i) + 2 \left( \frac{b - E - b}{E - 1} \right) p(i)^2 \right)$$  

(38)

and hence the noise distribution value for the pair $ii$ is

$$\xi(ii) = \frac{1}{2b - 1} \left( - \frac{b - 1}{E - 1} p(i) + 2 \left( \frac{b - E - b}{E - 1} \right) p(i)^2 \right).$$  

(39)

We see that the noise distribution depends on the batch size $b$. This is not surprising: For example, if the batch size is equal to one, the ordered pair $ij$ can only be sampled as a negative sample in the single batch that consists of that pair. Indeed, for $b = 1$ our formula yields

$$\xi(ij) = p(ij).$$  

(40)
meaning that the data and the noise distributions coincide. Conversely, if \( b = E \) and there is only one batch, we obtain

\[
\xi(ij) = \frac{2E}{2E - 1} p(i)p(j)
\]

(41)

and the noise distribution is close to uniform. For batch sizes between 1 and \( E \) the noise distribution is in between these two extremes. For MNIST, \( E \approx 1.5 \cdot 10^6 \), and in our experiments we used \( b = 1024 \). This means that the prefactor of the share of the data distribution is about 0.0005 while that of the near-uniform distribution \( p(i)p(j) \) is about 0.9995, so the resulting noise distribution is close to uniform. Note that Thm. [1] and Cor. [2] only require the noise distribution to be nonzero where the data distribution is nonzero, which is the case for any batch size.

### D Optimization tricks in UMAP’s reference implementation

UMAP’s repulsive term

\[
-\log(1 - \phi(ij)) = \log \left( \frac{1 + d_{ij}^2}{d_{ij}^2} \right)
\]

(42)

can lead to numerical problems if the two points of the negative sample pair are very close. In addition to the learning rate decay, discussed in Sec. [6] UMAP’s implementation uses further tricks to prevent unstable or even crashing training.

In non-parametric UMAP’s reference implementation, the gradient on embedding position \( e_i \) exerted by a single sampled repulsive pair \( ij \) is actually

\[
2 \frac{1}{d_{ij}^2 + \zeta} \frac{1}{1 + d_{ij}^2} (e_j - e_i)
\]

(43)

with \( \zeta = 0.001 \) instead of \( \zeta = 0 \). This corresponds to the full loss function

\[
-\mathbb{E}_{ij \sim p} \log(\phi(ij)) - m \left( 1 + \frac{\zeta}{1 - \zeta} \right) \mathbb{E}_{ij \sim \xi} \log \left( 1 + \frac{\zeta}{1 - \zeta} - \phi(ij) \right).
\]

(44)

However, we found that \( \zeta \) does not have much influence on the appearance of a UMAP embedding. Fig. [SI] shows MNIST embeddings obtained using the original UMAP implementation modified to use different values of \( \zeta \). Neither a much smaller positive value such as \( \zeta = 10^{-10} \) nor setting \( \zeta = 0 \) substantially changed the appearance of the embedding (even though some runs with \( \zeta = 0 \) did crash). The learning rate annealing played a much bigger role in how the embedding looked like (Fig. [SI] bottom row).

The reference implementation of parametric UMAP uses automatic differentiation instead of implementing the gradients manually. To avoid terms such as \( \log(0) \) in the repulsive loss, it clips the argument of the logarithm from below at the value \( \epsilon = 10^{-4} \), effectively using the loss function

\[
-\mathbb{E}_{ij \sim p} \log \left( \max \left\{ \epsilon, \frac{1}{1 + d_{ij}^2} \right\} \right) - m \mathbb{E}_{ij \sim \xi} \log \left( \max \left\{ \epsilon, 1 - \frac{1}{1 + d_{ij}^2} \right\} \right).
\]

(45)

We employ a similar clipping in our code whenever we apply the logarithm function. Again, we found that the exact value of \( \epsilon \) is not important for our UMAP implementation, while using the learning rate annealing is (Fig. [S2] top two rows). In the extreme case of setting \( \epsilon = 0 \) our UMAP runs crashed. We believe that the reason is that we allow negative sample pairs to be of the form \( ii \), which would not send any gradient, but would lead to a zero argument to the logarithm. The reference implementation of UMAP excludes such negative sample pairs \( ii \).

Our Neg-\( t \)-SNE approach does not have any of these problems, as the repulsive term is

\[
-\log \left( 1 - \frac{1}{1 + d_{ij}^2} \right) = \log \left( \frac{2 + d_{ij}^2}{1 + d_{ij}^2} \right) \leq \log(2)
\]

(46)

and does not diverge for \( d_{ij} \to 0 \). For this reason, Neg-\( t \)-SNE is not very sensitive to the value at which we clip arguments to the logarithm and works even with \( \epsilon = 0 \), both with and without learning rate annealing (Fig. [S2] bottom two rows).
Figure S1: UMAP embeddings of the MNIST dataset, ablating numerical optimization tricks of UMAP’s reference implementation. The learning rate annealing is crucial (bottom row) but safeguarding against divisions by zero in UMAP’s repulsive term (43) by adding $\zeta$ to the denominator has little effect. These experiments were run using the reference implementation, modified to change the $\zeta$ value and/or to switch off the learning rate annealing.

The attractive terms in the loss functions do not pose numerical problems in practice due to the heavy tail of the Cauchy kernel. That said, in order to keep different experiments and losses comparable, we used clipping in both the attractive and the repulsive loss terms with $\varepsilon = 10^{-10}$ for all neighbor embedding plots computed with our framework unless otherwise stated.

A recent pull request\(^4\) to the parametric part of UMAP’s reference implementation proposed another way to ameliorate the numerical instabilities. The clipping of the arguments to the logarithm was replaced with a sigmoid of the logarithm of the Cauchy kernel, so that the attractive and repulsive terms become

\[
-\log \left( \max \{ \varepsilon, \phi(ij) \} \right) \to -\log \left( \sigma \left( \log \left( \phi(ij) \right) \right) \right)
\]

\[
-\log \left( \max \{ \varepsilon, 1-\phi(ij) \} \right) \to -\log \left( 1-\sigma \left( \log \left( \phi(ij) \right) \right) \right),
\]

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function. This change can seem drastic as, e.g., for $\phi(ij) = 1$ we have $\max \{ \varepsilon, \phi(ij) \} = 1$, but $\sigma \left( \log (\phi(ij)) \right) = 1/2$. But unravelling the definitions shows that this turns the loss function precisely into our Neg-t-SNE loss function since

\[
\sigma \left( \log (\phi(ij)) \right) = \frac{1}{1 + \exp \left( -\log (\phi(ij)) \right)} = \frac{1}{1 + \phi(ij) - 1} = \frac{\phi(ij)}{\phi(ij) + 1}.
\]

So, in order to overcome the numerical problems incurred by UMAP’s implicit choice of $1/d_{ij}^2$ as similarity kernel, the pull request suggested a fix that turns out to be equivalent to negative sampling using the Cauchy kernel. We encourage this change to UMAP as it makes its loss function equivalent to our Neg-t-SNE and thus also conceptually more related to $t$-SNE. We also suggest to implement it in the non-parametric case.

E Datasets

We used the well-known MNIST \(^{30}\) dataset for most of our experiments. We downloaded it via the torchvision API from \url{http://yann.lecun.com/exdb/mnist/} This website does not

\[^4\]https://github.com/lmcinnes/umap/pull/856
Figure S2: UMAP and Neg-t-SNE embeddings of the MNIST dataset using different values $\varepsilon$ at which we clip arguments to logarithm functions. These experiments were done using our implementation. Varying $\varepsilon$ did not strongly influence the appearance of the embedding. But setting $\varepsilon = 0$ led to crashing UMAP runs. Annealing the learning rate is important for UMAP, yet not for Neg-t-SNE.

give a license. But https://keras.io/api/datasets/mnist/ and http://www.pymvpa.org/datadb/mnist.html name Yann LeCun and Corinna Cortes as copyright holders and claim MNIST to be licensed under CC BY-SA 3.0, which permits use and adaptation. The MNIST dataset consists of 70,000 grayscale images, 28 $\times$ 28 pixels each, that show handwritten digits.

The SimCLR experiments were performed on the CIFAR-10 [28] dataset, another standard machine learning resource. We downloaded it via the sklearn.datasets.fetch_openml API from https://www.openml.org/search?type=data&sort=runs&id=40927&status=active. Unfortunately, we were not able to find a license for this dataset. CIFAR-10 consists of 60,000 images, 32 $\times$ 32 RGB pixels each, depicting objects from five animal and five vehicle classes.

The transcriptomic dataset of Kanton et al. [23] was downloaded from https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-7552/ which permits free use. We only used the 20,272 cells in the human brain organoid cell line ‘409b2’. The downloaded UMIs were preprocessed as in [5, 23]. After selecting the 1000 most variable genes, we normalized the library sizes to the median library size in the dataset, log-transformed the normalized values with $\log_2(x+1)$, and finally reduced the dimensionality to 50 via PCA.
We mentioned in Sec. 5 and showed in Fig. S4 that one can move along the attraction-repulsion spectrum also by changing the number of noise samples $m$, instead of the fixed normalization constant $\bar{Z}$. In UMAP’s reference implementation, there is a scalar prefactor $\gamma$ for the repulsive forces. Theoretically, adjusting $\gamma$ should also move along the attraction-repulsion spectrum, but setting it higher than 1 led to convergence problems in [5], Fig. A11. When varying our $\bar{Z}$, we did not have such issues.

All contrastive embeddings were computed with our PyTorch [40] implementation of Neg-$t$-SNE, NCVis, UMAP, and InfoNC-$t$-SNE. Exceptions are Fig. 1 and analogous Figs. S7 and S8. There for panel [10] we used the reference implementation of NCVis [2] (with a fixed number of noise samples $m$, and not the default schedule), and for panel [21] we used UMAP 0.5. The $t$-SNE plots were created with the openTSNE [42] (version 0.6.1) package. Similarly, we used the reference UMAP implementation in Fig. S1 and openTSNE in Figs. S9 and S10.

We extended these implementations of NCVis, UMAP, and $t$-SNE to make them accept custom embedding initializations and unweighted skNN graphs and to log various quantities of interest. We always used the standard Cauchy kernel for better comparability.

All PCAs were computed with sklearn [41]. We used PyKeOps [8] to compute the exact skNN graph and to handle the quadratic complexity of computing the partition functions on a GPU. The same PCA initialization and skNN graph with $k = 15$ were used for all embeddings. The skNN graph for MNIST was computed on a 50-dimensional PCA of the dataset.

When computing logarithms during the optimization of neighbor embeddings, we clip the arguments to the range $[10^{-10}, 1]$, save for Fig. S2, where we ablated this lower bound. The lower bound is smaller than in the reference implementation of parametric UMAP, where it is set to $10^{-4}$.

Our defaults were a batch size of 1024, linear learning rate annealing from 1 (non-parametric) or 0.001 (parametric) to 0 (save for Figs. 2 [S1] and S2), 750 epochs (save for Fig. S6) and $m = 5$ noise samples (save for Figs. S4 [S9] and S10).

We initialized all embeddings with a scaled version of PCA (save for in Figs. S9 and S10). For $t$-SNE embeddings we rescaled the initialization so that the first dimension has a standard deviation of 0.0001 (as is default in openTSNE), for all other embeddings to a standard deviation of 1.

We employed some version of ‘early exaggeration’ [54] for the first 250 epochs in most non-parametric plots. For $t$-SNE it is the default early exaggeration of openTSNE. When varying $\bar{Z}$ in non-parametric Neg-$t$-SNE, early exaggeration meant using $\bar{Z} = |X|/m$ for the first 250 epochs (save for Fig. S8). When varying the number of noise samples in Figs. S4 [S9] and S10, we still used $m = 5$ for the first 250 epochs. In Figs. 2 [S1] and S2 as well as in all reference NCVis or UMAP plots, we did not use early exaggeration as neither small $\bar{Z}$ nor high $m$ made it necessary. When we used some form of early exaggeration and learning rate annealing, the annealing to zero took place over the first 250 epochs, was then reset, and annealed again to zero for the remaining, typically 500, epochs.

Non-parametric runs were optimized with SGD without momentum and parametric runs with the Adam optimizer [24]. Parametric runs used the same feed-forward neural net architecture as the reference parametric UMAP implementation. That is, four layers with dimensions input dimension — 100 — 100 — 100 — 2 with ReLU activations in all but the last one. We used the vectorized, 786-dimensional version of MNIST as input to the parametric neighbor embedding methods (and not the 50-dimensional PCA; but the skNN graph was computed in the PCA space for consistency with non-parametric embeddings).

Like the reference NCVis implementation, we used the fractions $q_{0,Z}(x)/(q_{0,Z}(x) + m)$ instead of $q_{0,Z}(x)/(q_{0,Z}(x) + m \xi(x))$. This is a mild approximation as the noise distribution is close to uniform. But it means that the model learns a scaled data distribution (cf. Cor. 2), so that we need to multiply the learned normalization parameter $Z$ by $n(n - 1)$ when comparing to $t$-SNE or checking normalization of the NCVis model. Similarly, we also approximate the true noise distribution by the uniform distribution for the fractions $q_{0}(x)/(q_{0}(x) + \bar{Z} m/|X|)$ — instead of $q_{0}(x)/(q_{0}(x) + \bar{Z} m \xi(x))$ — in our Neg-$t$-SNE implementation.

We mentioned in Sec. 3 and showed in Fig. S4 that one can move along the attraction-repulsion spectrum also by changing the number of noise samples $m$, instead of the fixed normalization constant $\bar{Z}$. In UMAP’s reference implementation, there is a scalar prefactor $\gamma$ for the repulsive forces. Theoretically, adjusting $\gamma$ should also move along the attraction-repulsion spectrum, but setting it higher than 1 led to convergence problems in [5], Fig. A11. When varying our $\bar{Z}$, we did not have such issues.
We ran our neighbor embedding experiments on a machine with 56 Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 502 GB RAM and 10 NVIDIA TITAN Xp GPUs. The SimCLR experiments were run on a machine with 56 Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 502 GB RAM and 10 NVIDIA TITAN Xp GPUs.

### F.1 Stability

Whenever we reported a metric or show a graph, we ran the experiments for 3 different random seeds, and reported the mean ± the standard deviation. When the standard deviation was very small, we omitted it from the main text and report it here. The normalization parameter $Z^{NCVis}$ learned by NCVis on the MNIST dataset is $(3.43 ± 0.01) \cdot 10^7$ and t-SNE does not depend on a random seed save for the usually approximate $k$NN graph computation. As we computed the $k$NN exactly with PyKeOps [8], t-SNE is deterministic in our framework. The values of the partition function for t-SNE on MNIST for Figs. S9h and S9i were $6.25 \cdot 10^6$ and $8.13 \cdot 10^6$ without and with early exaggeration, respectively. On the transcriptomics dataset, the learnt normalization parameter of NCVis was $(3.57 ± 0.03) \cdot 10^7$ and t-SNE’s partition function was $1.3 \cdot 10^6$.

Panels [I] in Figs. [I] S7 and S8 show the standard deviation as shaded area. Again, the standard deviations are very small and barely visible. The ratio of standard deviation to mean was never larger than $0.006$ in panels [I] of Figs. [I] S7 and S8. Similarly, the standard deviation in Figs. S5 and S6 shown as shaded area, is mostly smaller than the line width.

|                        | Number of runs | Time per run [min] (mean±SD) |
|------------------------|----------------|-------------------------------|
| Neg-t-SNE for Figs. [I] S7 S8 | 216            | 39 ± 4                        |
| NCVis (our implementation) for Fig. S6b | 3              | 786 ± 2                      |
| SimCLR runs for Tab. [I] | 12             | 694 ± 28                     |

For panels [I] in Figs. [I] S7 and S8 the Neg-t-SNE spectra were computed for $\hat{Z}$ equal to $Z(θ^{t-SNE})$, $Z^{NCVis}$, and $\frac{m(n-1)}{m} \cdot x$, where $x \in \{5 \cdot 10^{-5}, 1 \cdot 10^{-4}, 2 \cdot 10^{-4}, 5 \cdot 10^{-4}, \ldots, 1 \cdot 10^2, 2 \cdot 10^2, 5 \cdot 10^2\}$.

For the SimCLR experiments, we clipped arguments to logarithm functions at the default value $10^{-4}$. The learning rate was set to $0.12 = 0.03 \cdot \text{batch size}/256$, as done in [9]. Furthermore, we trained the model for 1000 epochs, of which we used 10 epochs for warmup. The learning rate during warmup was linearly interpolated from 0 to the initial learning rate. After the warmup epochs, we annealed the learning rate with a cosine schedule (without restarts) to 0.00012 [34]. We optimized the model parameters with SGD and momentum 0.9. We used the same data augmentations as in [9]. We used a ResNet18 [17] as the backbone and a projection head consisting of four layers with dimensions $512 - 100 - 100 - 100 - 128$. The losses were applied to the $L_2$ normalized output of the projection head, but like Chen et al. [9] we used the output of the ResNet as the representation for the linear evaluation. As similarity function we used the exponential of the normalized scalar product (cosine similarity) and always kept the temperature at 0.5, as suggested in Chen et al. [9]. When considering the entire batch as negative samples, we omit both head and tail of the considered positive sample, while we only omit the head when sampling a smaller number of negative samples.

The ResNet was trained on the combined CIFAR-10 train and test sets. When training the classifier, we froze the ResNet and only used the train set. The reported metrics are computed on the test set. We chose sklearn’s KNearestNeighbors classifier with cosine metric and $k = 15$ neighbors and sklearn’s SGDClassifier. For the linear classifier with augmentations, we followed Chen et al. [9] and randomly flipped the images as well as randomly resized the images. The linear classifier was a simple linear layer that maps from $512$ to the $10$ classes and was trained via a cross-entropy loss. The classifier was trained for $100$ epochs and an initial learning rate of $0.01$ that was annealed with a cosine schedule to $0$. We warmed up the batch normalization layers [20] by iterating over the data set five times and passing the samples through the backbone (discarding the result and not performing any backpropagation) before starting the training procedure of the linear classifier. Without warm-starting the batch normalization layers, we got much worse accuracies.

Our code is publicly available at [https://github.com/hci-unihd/cl-tsne-umap](https://github.com/hci-unihd/cl-tsne-umap).

### F.2 Compute

We ran our neighbor embedding experiments on a machine with 56 Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 502 GB RAM and 10 NVIDIA TITAN Xp GPUs. The SimCLR experiments were run on a machine with 56 Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 502 GB RAM and 10 NVIDIA TITAN Xp GPUs.
conducted on a Slurm cluster node with 8 cores of an Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz and a Nvidia V100 GPU with a RAM limit of 54.3 GB. Each experiment used one GPU at most.

Our total compute is dominated by the neighbor embedding runs for Figs. 1, S7, S8, and S6b and by the SimCLR experiments. Tab. S1 lists the number of runs and the average run time. We thus estimate the total compute time to be about 320 hours.

Our implementation uses pure PyTorch and relies on GPUs. But for the non-parametric experiments, much of the computation consists of sampling negative neighbors and computationally light updates to the embedding positions. Therefore, the CPU-based numba or C++ reference implementations of UMAP and NCVis are much faster. However, our implementation is arguably easier to inspect and adapt by the machine learning community and seamlessly integrates non-parametric and parametric settings as well as all four contrastive loss functions. It is thus more suited as a research tool rather than for large throughput application.

As the change from NCVis to Neg-t-SNE is as simple as fixing the learnable normalization parameter to a constant, we have also adapted the original NCVis code to compute Neg-t-SNE. We have not used this for any of the experiments of the paper, but include it as part of the supplementary as this implementation would retain the speed of the original NCVis implementation.

G Potential negative societal impact

Our work analyses connections between popular contrastive learning methods and in particular visualization methods. Visualization methods are fundamental in exploratory data analysis, e.g., in computational biology. Given the basic nature of visualization it can of course potentially be used for malicious goals, but we expect mostly positive societal effects such as more reliable exploration of biological data.

Similarly, contrastive self-supervised learning has the potential to overcome the annotation bottleneck that machine learning faces. This might free countless hours of labelling for more productive use but could also enable institutions with sufficient resources to learn from larger and larger datasets, potentially concentrating power.

Our PyTorch implementation of contrastive neighbor embeddings is slower than existing UMAP and NCVis implementations and hence more resource intensive. We therefore recommend it primarily as a research tool into contrastive learning and contrastive neighbor embeddings, and suggest more efficient implementations for large scale applications.
Algorithm S1: Batched contrastive neighbor embedding algorithm

input:
- list of directed kNN graph edges \( E = \{i_{1j}, \ldots, i_{|E|j}\}\)
- parameters \( \theta \) // embeddings (non-param.)/ network weights (param.)
- number of epochs \( T \)
- learning rate \( \eta \)
- number of noise samples \( m \)
- Cauchy kernel \( q // \) of embeddings (non-param.)/ network output (param.)
- batch size \( b \)
- loss mode \( \text{mode} \)
- normalization constant \( \bar{Z} \) // default \(|E|/m\), required for \( \text{mode} = \text{Neg-t-SNE} \)

output:
- final embeddings \( e_1, \ldots, e_n \)

1. if \( \text{mode} = \text{NCVis} \) then
   2. \( Z = 1 \)
   3. for \( t = 0 \) to \( T \) do
      4. // Learning rate annealing
         5. \( \eta_t = \eta \cdot (1 - \frac{t}{T}) \)
         6. \( \alpha = 0 \)
      7. while \( \alpha < |E| \) do
         8. \( \mathcal{L} = 0 \)
         9. for \( \beta = 1, \ldots, b \) do
            10. // Sample noise edge tails but omit head of considered edge
                11. \( j^{-}_1, \ldots, j^{-}_m \sim \text{Uniform}(i_{\alpha+1}, j_{\alpha+1}, \ldots, i_{\alpha+b}, j_{\alpha+b}) \)
            12. // Aggregate loss based on mode
               13. if \( \text{mode} = \text{Neg-t-SNE} \) then
                  14. \( \mathcal{L} = \mathcal{L} - \log \left( \frac{q_\theta(i_{\alpha+\beta} j_{\alpha+\beta})}{q_\theta(i_{\alpha+\beta} j_{\alpha+\beta}) + \bar{Z} m / |E|} \right) - \sum_{\mu=1}^{m} \log \left( \frac{q_\theta(i_{\alpha+\beta} \hat{j}_\mu)}{q_\theta(i_{\alpha+\beta} j_{\alpha+\beta}) + \bar{Z} m / |E|} \right) \)
               15. else if \( \text{mode} = \text{NCVis} \) then
                  16. \( \mathcal{L} = \mathcal{L} - \log \left( \frac{q_\theta(i_{\alpha+\beta} \hat{j}_\mu)/Z}{q_\theta(i_{\alpha+\beta} \hat{j}_\mu)/Z + m} \right) - \sum_{\mu=1}^{m} \log \left( \frac{q_\theta(i_{\alpha+\beta} \hat{j}_\mu)/Z}{q_\theta(i_{\alpha+\beta} j_{\alpha+\beta})/Z + m} \right) \)
               17. else if \( \text{mode} = \text{InfoNC-t-SNE} \) then
                  18. \( \mathcal{L} = \mathcal{L} - \log \left( \frac{q_\theta(i_{\alpha+\beta} \hat{j}_\mu)}{q_\theta(i_{\alpha+\beta} \hat{j}_\mu) + \sum_{\mu=1}^{m} q_\theta(i_{\alpha+\beta} j_{\alpha+\beta})} \right) \)
               19. else if \( \text{mode} = \text{UMAP} \) then
                  20. \( \mathcal{L} = \mathcal{L} - \log \left( q_\theta(i_{\alpha+\beta} \hat{j}_\mu) \right) - \sum_{\mu=1}^{m} \log \left( q_\theta(i_{\alpha+\beta} j_{\alpha+\beta}) \right) \)
            11. // Update parameters with SGD (non-param.) or Adam (param.)
               12. \( \theta = \theta - \eta_t \cdot \nabla_{\theta} \mathcal{L} \)
               13. if \( \text{mode} = \text{NCVis} \) then
                  14. \( Z = Z - \eta_t \nabla_{Z} \mathcal{L} \)
                  15. \( \alpha = \alpha + b \)
                  16. Shuffle \( E \)
               21. return \( \theta \)
Additional figures

Figure S3: Attractive and repulsive loss terms of UMAP and Neg-t-SNE. The main difference is that UMAP’s repulsive loss diverges at zero challenging its numerical optimization. The attractive terms are \( \log(1 + d_{ij}^2) \) and \( \log(2 + d_{ij}^2) \) for UMAP and Neg-t-SNE, respectively, and the repulsive ones are \( \log\left(\frac{1 + d_{ij}^2}{d_{ij}^2}\right) \) and \( \log\left(\frac{2 + d_{ij}^2}{1 + d_{ij}^2}\right) \), respectively.

Figure S4: Neg-t-SNE embeddings of the MNIST dataset for varying number of noise samples \( m \) and using batch size \( b = 1024 \). While for NCVis and InfoNC-t-SNE more noise samples improve the approximation to t-SNE, see Figs. S9 and S10, changing \( m \) in Neg-t-SNE moves the result along the attraction-repulsion spectrum (Fig. 1) with more repulsion for larger \( m \). However, the computational complexity of Neg-t-SNE scales with \( m \), so that moving along the spectrum via changing \( \bar{Z} \) is much more efficient. For the first 250 epochs, \( m \) was set to 5, to achieve an effect similar to early exaggeration (Appendix H).
Figure S5: (a, b) Loss curves for the parametric and non-parametric InfoNC-t-SNE and NCVis optimizations leading to Figs. 3b, c, e, and f. While the embedding scale differs drastically between the non-parametric and the parametric run, the loss values are close. (c) Normalization of the model $\sum_{ij} \phi(ij)/Z$ for the parametric and non-parametric NCVis optimizations. The difference in the embedding scale is compensated by a three orders of magnitude change in $Z$, so that both versions learn approximately normalized models. These experiments used our NCVis reimplementation.

Figure S6: NCVis learns to have the same partition function (PF) as t-SNE on the MNIST dataset. The higher the number $m$ of noise samples or the longer the optimization, the better the match. Both methods used early exaggeration, which for NCVis meant to start with $m = 5$ noise samples for the first 250 epochs. The learnt normalization parameter $Z$ converged to but did not exactly equal NCVis’ partition function $\sum_{ij} q_\theta(ij)$, but was of the same order of magnitude. Again, the match was better for more noise samples. Since we reinitialized the learnable $Z$ for NCVis after the early exaggeration phase, there were brief jumps in the partition function and in $Z$ at the beginning of the non-exaggerated phase.
Figure S7: (a–e) Neg-t-SNE spectrum on the developmental single-cell RNA sequencing dataset from [23] for various parameters \( \bar{Z} \). As \( \bar{Z} \) increases, the scale of the embedding decreases and the continuous structure (corresponding to the developmental stage) becomes more apparent, making higher \( \bar{Z} \) more suitable for visualizing continuous datasets [5]. The spectrum produces embeddings very similar to those of (f) t-SNE and (g) NCVis when \( \bar{Z} \) equals the partition function of t-SNE or the learned normalization parameter of NCVis. The UMAP embedding in (h) closely resembles the Neg-t-SNE embedding at \( \bar{Z} = |X|/m = (n^2)/m \). (i) The partition function \( \sum_{ij} (1 + d_{ij}^2)^{-1} \) tries to match \( \bar{Z} \) and grows with it. In contrast to Fig. 1, we did not use early exaggeration here, but initialized the Neg-t-SNE and t-SNE with PCA rescaled so that the first dimension has standard deviation 1 and 0.0001, respectively. This makes the embeddings with small \( \bar{Z} \) values show cluster fragmentation, similar to the t-SNE embedding in (f) without early exaggeration. For very low \( \bar{Z} \), the Neg-t-SNE embedding in (a) shows very little structure.

Figure S8: (a–e) Neg-t-SNE embeddings of the MNIST dataset for various values of the fixed normalization constant \( \bar{Z} \). As \( \bar{Z} \) increases, the scale of the embedding decreases, clusters become more compact and separated before eventually starting to merge. The Neg-t-SNE spectrum produces embeddings very similar to those of (f) t-SNE, (g) NCVis, and (h) UMAP, when \( \bar{Z} \) equals the partition function of t-SNE, the learned normalization parameter \( Z \) of NCVis, or \( |X|/m = (n^2)/m \) used by UMAP, as predicted in Sec. 4–6. (i) The partition function \( \sum_{ij} (1 + d_{ij}^2)^{-1} \) tries to match \( \bar{Z} \) and grows with it. In contrast to Fig. 1, we did not use early exaggeration here, but initialized the Neg-t-SNE and t-SNE with PCA rescaled so that the first dimension has standard deviation 1 and 0.0001, respectively. This makes the embeddings with small \( \bar{Z} \) values show cluster fragmentation, similar to the t-SNE embedding in (f) without early exaggeration. For very low \( \bar{Z} \), the Neg-t-SNE embedding in (a) shows very little structure.
Figure S9: NCVis (our implementation) on the MNIST dataset for varying number of noise samples $m$ and different starting conditions. Higher number of noise samples $m$ improves the approximation quality to t-SNE (last column). The first row is initialized with isotropic Gaussian noise and the second and the third rows with PCA (both normalized to have standard deviation of one or 0.0001 in the first dimension for NCVis or t-SNE, respectively). In the third row, the first 250 epochs used $m = 5$ and the latter used the given $m$ value for NCVis. This is similar to t-SNE’s early exaggeration that we used in panel l. NCVis seems to be less dependent on early exaggeration than t-SNE, especially for low $m$ values.
Figure S10: InfoNC-t-SNE on the MNIST dataset for varying number of noise samples $m$ and different starting conditions. Higher number of noise samples $m$ improves the approximation quality to $t$-SNE (last column). The first row is initialized with isotropic Gaussian noise and the second and the third rows with PCA (both normalized to have standard deviation of one or 0.0001 in the first dimension for InfoNC-t-SNE or $t$-SNE, respectively). In the third row, the first 250 epochs used $m = 5$ and the latter used the given $m$ value for InfoNC-t-SNE. This is similar to $t$-SNE’s early exaggeration that we used in panel (l). InfoNC-t-SNE seems to be less dependent on early exaggeration than $t$-SNE, especially for low $m$ values.