DOI: 10.1111/cgf.14834

Eurographics Conference on Visualization (EuroVis) 2023
D. Archambault, R. Bujack, and T. Schreck
(Guest Editors)

COMPUTER GRAPHICS *forum*
*Volume 42 (2023), Number 3*

# ParaDime: A Framework for Parametric Dimensionality Reduction

Andreas Hinterreiter[1], Christina Humer[1], Bernhard Kainz[2,3], and Marc Streit[1]

[1]Johannes Kepler University Linz, Austria
[2]Friedrich-Alexander-University Erlangen-Nuremberg, Germany
[3]Imperial College London, UK

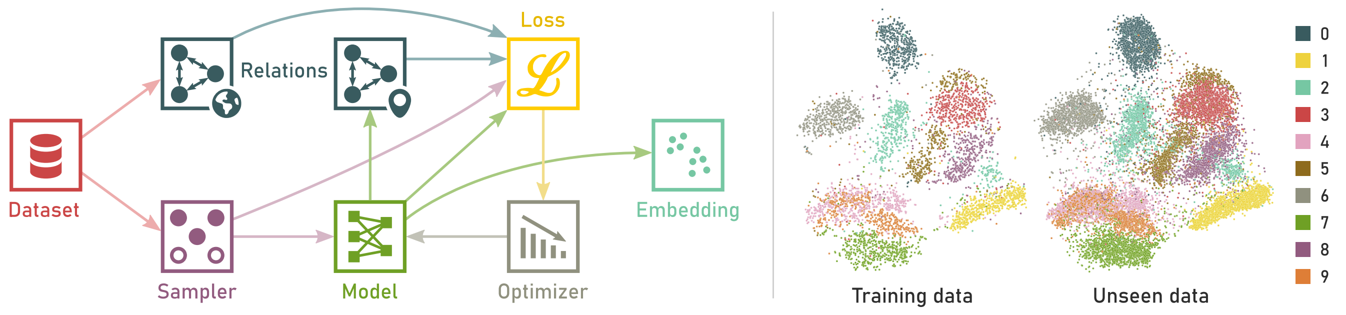

**Figure 1:** *ParaDime is a framework for parametric dimensionality reduction. Left: Data flow in a single training phase of a ParaDime routine. Right: Parametric t-SNE trained on a subset of 5000 images from the MNIST dataset [LeC05] and applied to 15,000 unseen images.*

**Abstract**

*ParaDime is a framework for parametric dimensionality reduction (DR). In parametric DR, neural networks are trained to embed high-dimensional data items in a low-dimensional space while minimizing an objective function. ParaDime builds on the idea that the objective functions of several modern DR techniques result from transformed inter-item relationships. It provides a common interface for specifying these relations and transformations and for defining how they are used within the losses that govern the training process. Through this interface, ParaDime unifies parametric versions of DR techniques such as metric MDS, t-SNE, and UMAP. It allows users to fully customize all aspects of the DR process. We show how this ease of customization makes ParaDime suitable for experimenting with interesting techniques such as hybrid classification/embedding models and supervised DR. This way, ParaDime opens up new possibilities for visualizing high-dimensional data.*

**CCS Concepts**

• ***Computing methodologies*** → ***Neural networks;*** *Learning latent representations;* • ***Human-centered computing*** → ***Visualization systems and tools; Information visualization;***

## 1. Introduction

Dimensionality reduction (DR) is one of the standard strategies for visualizing high-dimensional data. The general concepts of DR have been known and applied for over a century [AW10] in the form of linear techniques such as principal component analysis (PCA). In recent decades, however, nonlinear DR techniques have gained popularity. The most prominent modern techniques are t-distributed stochastic neighbor embedding (*t*-SNE) [vdMH08] and uniform manifold approximation and projection (UMAP) [MHM18]. Both *t*-SNE and UMAP rely on pairwise inter-item relationship information from high-dimensional data to construct embeddings in a low-dimensional space, with the goal of preserving key "structures" of the original data.

One shortcoming of such relationship-based DR techniques is that new items cannot readily be added to existing embeddings without recomputing all pairwise relationships. To address this shortcoming, researchers have developed *parametric* DR techniques. In parametric DR, embeddings are created by parameterized functions (e.g., neural networks) that are trained on high-dimensional data. While several implementations of parametric DR are available, most of them are tailor-made variations of existing techniques, and they are often difficult to customize or extend. This "scattered" nature of existing parametric DR techniques is surprising, considering the strong conceptual similarities between various nonlinear DR approaches [BBK22]. In particular, the loss in many DR techniques is based on the comparison of (transformed) pairwise distances between data points.

We believe that the potential of parametric DR is underexplored, and that both the visualization and the machine learning communities could benefit from a framework that makes it easy to experiment with such techniques. For this purpose, we introduce *ParaDime*, a unifying framework for parametric DR. Our contribution with ParaDime is threefold:

- We introduce a generalizing grammar which formalizes all the steps and building blocks necessary to specify a parametric DR routine.
- We show how this grammar can be used not only to create parametric versions of existing DR techniques, but also to experiment with new ideas.
- We present an implementation of the grammar with a focus on usability and customization.

Our paper is structured as follows: In Section 2 we summarize the historical development of DR, focusing on parametric techniques; in Section 3 we explain how similarities between these techniques give rise to a grammar of parametric DR, and how ParaDime implements this grammar; we then show how ParaDime can be used to create parametric versions of existing DR techniques (Section 4) and how it facilitates experimentation with new ideas (Section 5); in Section 6 we discuss design choices, ease of use, limitations, and future work; Section 7 concludes the paper.

## 2. Related Work

DR can be categorized broadly into linear and nonlinear techniques. The oldest linear technique, PCA, has been known for over a century [Pea01; AW10]. The survey by Cunningham and Ghahramani [CG15] provides an excellent overview of the many linear techniques that have been developed since PCA was introduced. Among these, multidimensional scaling (MDS) [Tor52] is most pertinent to our work. Classic MDS is an eigenvalue problem with a close relationship to PCA [Wil02; CC08]. In contrast, *metric* MDS is a more general approach that aims to find a low-dimensional configuration of points whose pairwise distances best match those of the high-dimensional data.

Metric MDS, with its principle of comparing pairwise distances, is the intellectual predecessor of many modern nonlinear techniques, such as Isomap [Ten00], SNE [HR02], *t*-SNE [vdMH08], and UMAP [MHM18]. Isomap tries to find a low-dimensional configuration based on geodesic (i.e., shortest-path) distances computed on a high-dimensional neighbor graph [Ten00]. In SNE, Gaussian kernels are used to transform pairwise distances into neighborhood probability distributions for both the high- and low-dimensional data. These probability distributions are then compared using the Kullback–Leibler (KL) divergence [HR02]. To avoid the so-called crowding problem in the resultant embeddings, *t*-SNE computes the probabilities in the low-dimensional space using the more fat-tailed Student's t-distribution [vdMH08]. Finally, UMAP replaces the t-distribution with a modified Cauchy distribution and uses a cross entropy loss instead of the KL divergence [MHM18; SMG21]. The conceptual similarities of these (and several more) nonlinear DR techniques were highlighted in various contexts by Bengio et al. [BDR*04], Böhm et al. [BBK22], and Agrawal et al. [AAB21]. Recently, the relationship between *t*-SNE and UMAP has been the subject of intense debate [BMH*19; KL21; DH21; DBHK22].

Aside from their conceptual similarities, many nonlinear DR techniques share a practical limitation: they involve the calculation of pairwise distances. Adding new points to existing embeddings—a problem known as out-of-sample extension—usually requires recomputing the whole embedding. This drawback has been addressed by approximating embeddings with parametric functions, using (*i*) kernel-based approaches [BDR*04; GMH12; GSH15], (*ii*) mixture models and data imputation [dBMVL19], or (*iii*) neural networks [vdMaa09; MvdMY*10; SMG21; LKL*22]. Parametric versions of *t*-SNE and UMAP are examples of manifold learning [BCV13], a subfield of representation learning. The general idea of using neural networks to reduce data dimensionality, in particular with autoencoders, predates these extensions [HZ93; Hin06]. Additionally, parametric nonlinear DR techniques based on neighborhood information are related to metric learning [Kul12], where representations are determined by learning a distance function.

Minimum distortion embeddings (MDEs) [AAB21] and the matrix optimization framework by Cunningham and Ghahramani [CG15] are closely related to our work in that they aim to unify several existing techniques in a common framework. Cunningham and Ghahramani view DR as a matrix optimization problem with varying objectives [CG15]. By choosing the right objective and/or matrix constraints, a wide variety of techniques can be expressed in their framework—albeit only *linear* ones. MDEs use formalized distortions and penalty functions to generalize non-linear embeddings [AAB21]. However, MDEs are nonparametric and support out-of-sample-extension only via a combination of anchoring constraints and solving a new MDE subproblem [AAB21]. In addition, MDEs are phrased in a way that makes it challenging to map them to existing techniques (see, e.g., the comparison of *t*-SNE and UMAP by Sainburg et al. [SMG21] vs. how Agarwal et al. relate penalties to UMAP [AAB21]). ParaDime focuses instead on (potentially transformed) pairwise relations between data items, which allows several existing techniques to be directly "translated" into its framework.

Furthermore, ParaDime uses neural networks to compute embeddings. As a result, well-established loss functions from other tasks, such as classification and reconstruction, can be readily included in ParaDime DR routines. This relates ParaDime to other techniques that add constraints to dimensionality reduction [VBF22]. In summary, ParaDime combines ideas from unifying nonlinear DR [BBK22; AAB21] with parametric DR [vdMaa09; SMG21], and provides flexibility to include alternative learning paradigms.

## 3. The ParaDime Grammar of Parametric DR

The similarities between the various neighbor- and distance-based DR techniques outlined above inspired us to develop a unifying interface for specifying parametric dimensionality reduction *routines*. In ParaDime, routines are complete data processing pipelines that include all the specifications necessary to generate a trained parametric DR model from a given dataset. In this section, we describe how routines can be specified with the ParaDime grammar of parametric DR. This approach follows the tradition of grammars and grammar-like structures in the visualization community, such as Vega [SRHH16], Vega-Lite [SMWH17] and Encodable [Won20] for general visualizations, Atom [PDFE18] for unit visualizations, Gosling [LWLG22] for genome visualizations, and Neo [GHM*22] for confusion matrices.

## 3.1. Overview

ParaDime generalizes parametric DR by breaking it down into several steps, as outlined in the data-flow graph in Figure 1. First, relations between all items in a given dataset are computed. Then, a batch of data is sampled in a training loop. The data batch is processed with a machine learning model, and new relations between all items in the processed batch are computed. The batch-wise relations are compared with an appropriate subset of the overall relations to compute an embedding loss. Additional losses may be added to the embedding loss. Finally, the losses are used to optimize the machine learning model.

The ParaDime grammar defines how the building blocks for each of these steps are specified. We use YAML for these specifications due to its focus on readability [dNMP*21]. A ParaDime specification requires the three base-level entries `relations`, `losses`, and `training phases`. Additionally, the `derived data` field may be used to specify how extra data should be computed from the dataset or the relations. In the following subsections, we explain each of these fields in detail. The model and dataset are not part of the specifications. They are provided separately by the user, as explained in Section 3.6.

## 3.2. Relations

The `relations` entry of a ParaDime specification lists “recipes” for computing mutual relations between data items. Each relation recipe is specified either globally or at the batch level. ParaDime computes `global` relations between all items in the dataset before any training begins; these are typically relations between the original, high-dimensional data points. In contrast, the computation of `batch`-wise relations is deferred to the training-loop stage of the routine. The `batch`-wise relations are computed between items in a batch of data that has been processed by the model (i.e., between the low-dimensional data points).

A relation’s `type` specifies how relations are computed; supported types are, for instance, exact pairwise distances (`pdist`) and approximate neighbor-based distances (`neighbor`). A relation’s `data` field specifies which part of the dataset to use to compute relations. ParaDime assumes that individual parts of a dataset can be accessed via keys, which are used as values for the `data` field. For example, a dataset might have its main data tensor and associated class labels stored under two different keys. Relations typically accept a set of `options`. For instance, distance-based relations allow users to specify the exact distance function to be used (e.g., `metric`: `euclidean`). Other relations allow algorithm-specific settings, such as the number of nearest neighbors for `neighbor`-based relations.

Finally, a list of `transforms` can be applied to the relations. Transforms can be used, for instance, to convert pairwise distances into perplexity-based probabilities of neighborhood as in *t*-SNE [vdMH08] (see Section 4.2). The complete `relations` specification has the following structure:

```
relations:
  - name: <rel name>
    level: global | batch
    type: <rel type>
    data: <data field to use>
    options: {...}
    transforms:
      - type: <transform type>
        options: {...}
      - ...
  - ...
```

Note that a routine can have any number of global or batch-wise relations. Each relation has a name so that it can be referenced by the `losses` or in `derived data`.

## 3.3. Losses

Once ParaDime knows how to compute relations between data items, these relations can be used within `losses` to construct objective functions that govern the training process. A ParaDime specification of a routine’s `losses` has the following structure:

```
losses:
  - name: <loss name>
    type: <loss type>
    func: <loss function>
    keys:
      data: [<data attr name>, ...]
      rels: [<rel name>, ...]
      methods: [<model method>, ...]
  - ...
```

Each loss has a `type`, which defines how it behaves during training. Supported loss types are `relation`, `classification`, `reconstruction`, and `position`. A loss of type `relation` compares a subset of precomputed global relations to relations computed from a processed batch of data. A `classification` loss compares the model output for a data batch to labels within the dataset. A `reconstruction` loss compares the original input batch to an encoded and decoded version of the batch. Finally, a `position` loss compares the low-dimensional output to a given set of coordinates. To retain flexibility, each loss includes a specification of the `keys` that should be used to access the relevant model methods, attributes of the data, and/or the relations. Losses can be combined during training to form weighted compound losses, as explained in the following subsection.

### 3.4. Training Phases

In ParaDime, the training of a routine is organized into `training phases`. Each training phase consists of `sampling` and `optimizer` specifications, a number of `epochs`, and a `loss` specification:

```
training phases:
  - epochs: <number of epochs>
    sampling:
      type: item | edge
      options: {...}
    loss:
      components: [<loss name>, ...]
      weights: [<weight>, ...]
    optimizer:
      type: <optim type>
      options: {...}
  - ...
```

The `sampling` type can be either `item` (simple sampling of batches of items) or `edge` (sampling of items based on relations between them). The `edge`-based sampling option enables ParaDime specifications of techniques that are based on negative-edge sampling [MSC*13; TLZM16; MHM18] or triplets [CSSB10] (see example in Section 5.2). As already mentioned above, the `loss` in each training phase is a weighted compound loss, whose `components` are specified with the names of the `losses` defined earlier. Finally, the `optimizer` entry specifies which optimization technique to use (e.g., `sgd` [Bot10] and `adam` [KB17]), along with options such as the learning rate or the momentum [SMDH13].

A ParaDime routine can have any number of training phases. Organizing the training into phases enables the pre-training of models, which can replace the initialization of low-dimensional positions used in non-parametric embeddings. It also allows multi-stage optimization schemes such as the early exaggeration often used in *t*-SNE [vdMH08].

### 3.5. Derived Data

As mentioned earlier, an optional `derived data` field in a ParaDime specification allows new dataset attributes that are populated right before training to be defined based on other data attributes or on global relations. They are specified as follows:

```
derived data:
  - name: <attr name>
    data func: <data function>
    keys: [[data | rels, <key>], ...]
  - ...
```

Here, the `keys` field allows users to specify which parts of the data or the relations are passed as arguments to the `data func` that computes the derived data. A simple use case for the `derived data` field would be the calculation of PCA for initialization purposes (see, e.g., the *t*-SNE example in Section 4.2). Our rephrasing of parametric UMAP in terms of ParaDime in Section 4.3 shows how derived entries can be used to set up initialization schemes based on transformed global relations.

### 3.6. Using the Grammar

ParaDime gives users two options for creating parametric DR routines. The first option is to parse YAML specifications as described above. In this case, users instantiate a ParaDime routine by loading a specification file and additionally passing a PyTorch [PGM*19] module as the model (i.e., neural network). ParaDime then parses the specification and sets up Python objects corresponding to the components specified. For each key in a specification, ParaDime allows only specific values that correspond to implemented classes or functions. If users want to parse specifications with custom values, these values and the corresponding implementations need to be registered beforehand (using ParaDime's registration methods). The second option is to set up the objects manually, using the ParaDime API rather than specification files. In this case, custom objects and functions can be used directly. Once users have instantiated a ParaDime routine, they can call its training method, passing the training data as an argument. Since PyTorch modules are typically initialized randomly, most ParaDime routines constitute random embeddings until the training method is called.

We provide a detailed documentation with examples and a less technical introduction of the building blocks of ParaDime routines online [Hin23a]. Paradime is pip-installable, and the code is available on GitHub [Hin23b].

## 4. Framing Existing Techniques in Terms of ParaDime

In this section, we show how (parametric extensions of) existing techniques can be specified in terms of the ParaDime grammar. Note that we omit the `weights` list in all cases, as all examples use only a single loss component per training phase.

### 4.1. Metric MDS

Metric multidimensional scaling aims to find a configuration of points in low-dimensional space such that the pairwise distances match those of the high-dimensional data [CG15]. This can be specified with ParaDime through Euclidean pairwise distance relations and a mean square error loss between the two relations:

```
relations:
  - name: dists hd
    level: global
    type: pairwise
    options:
      metric: euclidean
  - name: dists ld
    level: batch
    type: pairwise
    options:
      metric: euclidean
```

```
losses:
  - name: mds
    type: relation
    func: mse
    keys:
      rels:
        - dists hd
        - dists ld
training phases:
  - loss:
      components: mds
```

Figure 2 shows the normalized stresses [EMK*19] for several ParaDime routines with different models and the specification above trained on a 10-dimensional diabetes dataset [EHJT04]. The linear model was a simple matrix multiplication to map the 10-dimensional vectors to a 2-dimensional embeddings space. The nonlinear models were fully connected neural networks with hidden layer dimensions as indicated, an additional bias, and soft-

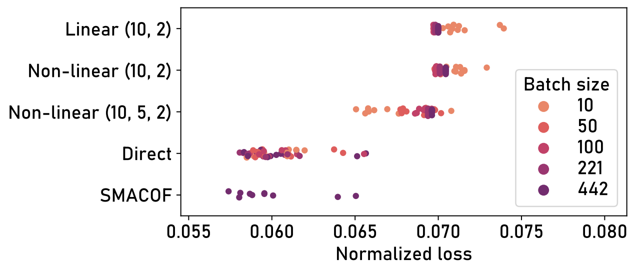

**Figure 2:** *Normalized stress [EMK*19] for parametric versions of metric MDS compared with the non-parametric SMACOF implementation of scikit-learn [PVG*11]. The non-linear models were fully connected neural networks with hidden layer dimensions as indicated. The routine labeled "Direct" is a non-parametric routine using a batch-wise optimization which mimics that of the parametric ones. All models were trained on a 10-dimensional diabetes dataset with 442 items [EHJT04].*

plus as activation function. The routine labeled *Direct* was a non-parametric routine implemented with ParaDime by replacing the model function with with a matrix that directly holds the embedding coordinates. All ParaDime routines used the same optimizer (Adam [KB17]), learning rate (0.01) and number of epochs (500). The losses of the ParaDime routines are compared with that of the non-parametric scikit-learn implementation using the SMACOF algorithm [Kru64]. Note how the routines with linear and nonlinear models of size $10 \times 2$ performed almost identically. Adding another hidden layer of dimension five reduced the loss substantially, especially for smaller batch sizes. The average loss for a batch size of ten was less than 12 % greater than the average of the SMACOF baseline, despite the simplicity of the model and the absence of hyperparameter tuning. Interestingly, for the two models of size $10 \times 2$, smaller batch sizes led to higher losses. The non-parametric implementation had losses similar to the SMACOF baseline. These results reveal the importance of model and hyperparameter selection, which we discuss in Section 6.4.

### 4.2. *t*-SNE

The *t*-SNE algorithm begins with calculating pairwise distances that are transformed into normalized and symmetrized probabilities of high-dimensional neighborhood based on a perplexity hyperparameter [vdMH08]. In low-dimensional space, probabilities of neighborhood are calculated by transforming Euclidean distances with a Student's t-distribution [vdMH08]. Defining these two relations in ParaDime using **`transforms`** is straightforward. Note that the global relation specification contains `neighbor` rather than `pdist` as **`type`**, which tells ParaDime to use approximate nearest-neighbor-based distances. This is an optimization that is used in modern *t*-SNE implementations [PSZ19]. The two probability matrices are compared using the KL divergence.

Before this step, most *t*-SNE implementations perform an initialization of the embedding with PCA coordinates. The embedding coordinates, however, cannot be initialized directly in a parametric DR routine, because the coordinates are outputs of a neural network. Instead, the model weights have to be set in such a way that the model mimics a PCA transformation. In ParaDime, this is achieved by pre-training the model in a separate training phase. The **`derived data`** specification makes the required PCA coordinates available during training. This results in the following ParaDime specification for parametric *t*-SNE:

```
derived data:
  - name: pca
    data func: pca
    keys: [[data, main]]
relations:
  - name: p
    level: global
    type: neighbor
    data: main
    options:
      metric: euclidean
    transforms:
      - type: perplexity
        options:
          perplexity: <p>
      - type: symmetrize
      - type: normalize
  - name: q
    level: batch
    type: pairwise
    data: main
    options:
      metric: euclidean
    transforms:
      - type: t-dist
```

```
        options:
          alpha: 1.
      - type: normalize
losses:
  - name: init
    type: position
    func: mse
    keys:
      data: [main, pca]
  - name: emb
    type: relation
    func: kl div
    keys:
      rels: [p, q]
training phases:
  # pca initialization
  - loss:
      components: [init]
      sampling:
        type: item
  # main embedding
  - loss:
      components: [emb]
      sampling:
        type: item
```

Parametric *t*-SNE as specified above does not feature early exaggeration [vdMH08]. However, this can easily be implemented by adding a training phase between the pre-training and embedding phases, making use of a simple multiplicative transform. In contrast to the parametric version of *t*-SNE recently introduced by Lai et al. [LKL*22], ParaDime currently does not use gradient clipping. In future version, gradient clipping could be included as an option in the loss specification.

An example of a parametric *t*-SNE routine implemented with ParaDime is shown in the right part of Figure 1. It was trained on a subset of 5000 images of the MNIST dataset of handwritten digits [LeC05] with a perplexity of 100 and a learning rate of 0.001. The model had hidden layer dimensions of 1024, 512, 256, 128 and used softplus for all activation functions. This model architecture is the same as the one used by Lai et al. [LKL*22], but our experiments suggest that models with far fewer parameters (e.g., hidden layer dimensions of 100 and 50) work reasonably well in many cases. Figure 1 also shows the result of applying the trained model to 15,000 unseen data instances.

### 4.3. UMAP

As discussed in Section 2, UMAP has several conceptual similarities to *t*-SNE. Its ParaDime specification therefore reads relatively similar to that of *t*-SNE. In the following, we omitted fields that are the same as in the specification for parametric *t*-SNE.

```
derived data:
  - ...
  - name: spectral
    data func: spectral
    keys: [[rels, p]]
relations:
  - ...
    transforms:
      - type: connect
        options:
          neighbors: <n>
      - type: symmetrize
        options:
          sub prod: true
      - type: normalize
  - ...
    transforms:
      - type: cauchy
        options:
          spread: <s>
          min dist: <md>
```

```
losses:
  - name: init
    type: position
    func: mse
    keys:
      data:
        - main
        - spectral
  - name: emb
    type: relation
    func: cross entropy
    keys:
      rels: [p, q]
training phases:
  # spectral init
  - ...
  # main embedding
  - loss:
      components: [emb]
      sampling:
        type: edge
```

Sainburg et al. [SMG21] outlined the main differences of UMAP from *t*-SNE. In contrast to *t*-SNE, UMAP:

- initializes coordinates with a `spectral` embedding based on global relations, instead of applying PCA;
- transforms distances to probabilities with kernels whose widths depend on `connectivity` instead of perplexity;
- transforms batch-wise relations with a modified `cauchy` distribution instead of a Student's t-distribution;
- uses `cross entropy` as loss instead of KL divergence; and
- uses negative-`edge` sampling instead of item-based sampling.

ParaDime uses an implementation of negative-edge sampling which does not ensure that each item is sampled at least once. This may lead to slightly smaller repulsive forces in ParaDime embeddings compared to an existing parametric UMAP version [SMG21].

The bottom four scatterplots in Figure 3 give an indication of how parametric UMAP embeddings look for the MNIST dataset [LeC05]. Note, however, that these embeddings come from routines with an additional loss term, as explained in Section 5.1.

### 4.4. Additional Neighbor-based Techniques

ParaDime includes implementations of all **`relations`**, **`transforms`**, and **`data func`** methods specified in the examples above. With these methods, it is also possible to specify LargeVis [TLZM16], which basically combines *t*-SNE's high-dimensional relations with negative-edge sampling. LargeVis is not restricted to a specific transform for the low-dimensional (i.e., `batch`-wise) relations; the authors state that "many probabilistic functions can be used" instead [TLZM16]. This aligns well with ParaDime's flexible concept of transforms.

Isomap is another neighbor-based technique, but it uses geodesic distances instead of probabilities of neighborhood [Ten00]. Specifying Isomap with ParaDime merely requires implementing either a new `relations` type or a `transform` that converts Euclidean distances to geodesic distances.

### 4.5. Classifiers & Autoencoders

In addition to the `relation`-type loss used in all DR techniques discussed so far, ParaDime also provides losses for typical machine-learning tasks that are not limited to DR. In particular, the `classification` loss makes it straightforward to implement classification models. The following specification assumes that the main data is accessible as `main`, and ground truth labels as `labels`.

```
losses:
  - type: classification
    func: cross entropy
    keys:
      data:
        - main
        - labels
```

Similarly, autoencoders can be concisely specified using the predefined `reconstruction` loss. Graving and Couzin [GC20], and Sainburg et al. [SMG21] have previously discussed the potential of combining the reconstruction ability of autoencoders with relation-based embedding losses.

## 5. Experimenting with Combined Techniques

In this section, we present several application ideas for ParaDime. These examples show the versatility of the ParaDime specifications, and encourage experimentation with new ideas that emerge from combining different losses.

### 5.1. Hybrid UMAP for Embedding and Classification

In Sections 4.3 and 4.5 we showed how to use ParaDime to specify a parametric version of UMAP and a simple classification model, respectively. In this section, we combine the two to create a hybrid embedding and classification routine which uses a shared latent space for both tasks. We applied our multitask routine to the MNIST dataset of handwritten digits [LeC05].

As a model, we used a fully connected network with hidden-layer dimensions 100 and 50. The model has two output layers: one of dimension ten that yields the logits used for classification, and one of dimension two for the embedding. Both these output layers are connected to the second hidden layer.

As explained above, UMAP uses edge-based sampling. When edge-based sampling is specified in ParaDime, each batch contains not only the pairs of vertices between the sampled edges, but also a list of unique data items suitable for other tasks, such as classification. Therefore, losses that require item-based sampling can readily be added to routines that use negative-edge sampling. The specification below creates our hybrid classification and embedding model, with previously defined losses and relations omitted.

```
relations: <UMAP relation specs>
losses: [<UMAP loss>, <classification loss>]
training phases:
  - loss:
      components: [umap, class]
      weights: <w>
```

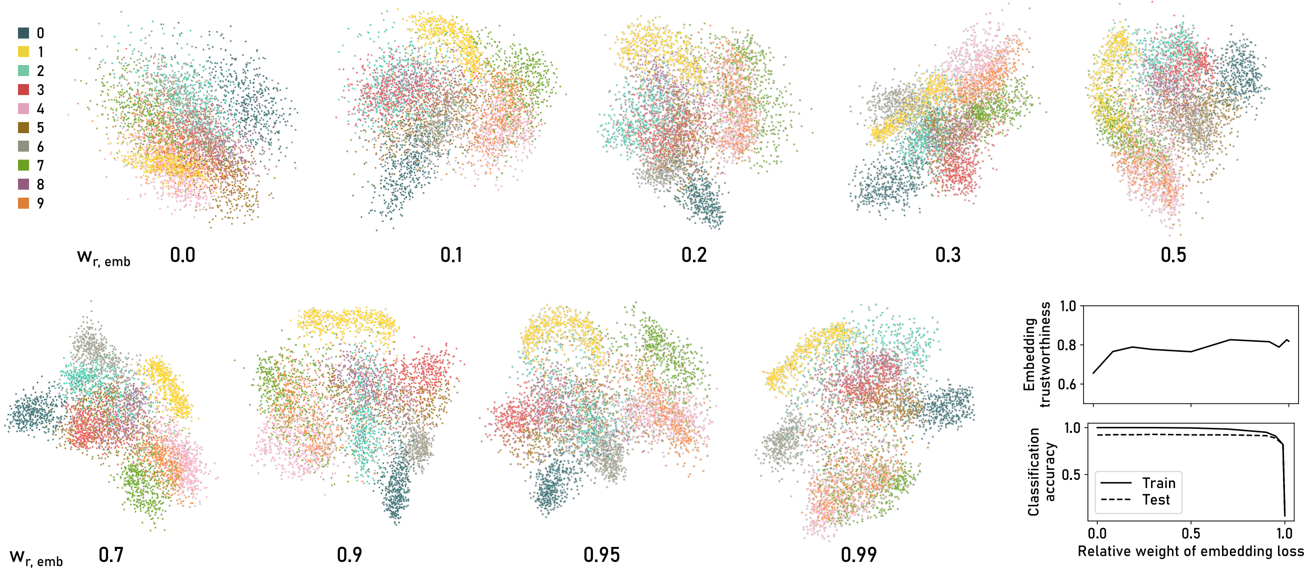

**Figure 3:** *Embeddings of hybrid embedding/classification routines for the MNIST dataset [LeC05] created with ParaDime. The relative weight of the embedding loss component is indicated by $w_{r,emb}$, and the weight of the classification component was $1 - w_{r,emb}$. All embedding-related specifications were the same as those of the ParaDime parametric UMAP routine. The routines were trained on a subset of 5000 randomly sampled MNIST images. Test accuracy was calculated on a different subset of 5000 images. Trustworthiness [VK01; EMK*19] was calculated based on ten nearest neighbors.*

Thanks to ParaDime's specification interface, the losses above can be simply reused as components in a compound loss. Figure 3 shows nine embeddings created with different weights for the loss components. All routines were trained on the same subset of 5000 images from MNIST for 100 epochs and without any pre-training. Figure 3 also includes plots of the classification accuracy and the embedding trustworthiness (as defined by Venna and Kaski [VK01; EMK*19]) as functions of the weight. The accuracy was calculated using a non-overlapping test subset of 5000 random images. Note that even a small weight on the embedding loss leads to a substantial class separation in the scatterplots. At the same time, classification accuracy is not affected by the additional embedding task. The accuracy suffers only when the weight on the classification approaches zero. Weighting the embedding with values in the wide range of 0.5 to 0.95 produces visually "sensible" embeddings with relatively high trustworthiness and practically the same classification accuracy as the pure classifier. In fact, some of our experiments showed that the additional embedding loss can slightly improve generalization of the classifier. This observation is in line with the original motivation for multitask learning [Car97].

Such a hybrid embedding and classification model could form the basis for a visualization tool in which users can add new points to existing embeddings. The predicted class labels could be used to visually encode the new data points and/or to inform users whether a new point lies within a region of the embedding where other points of the same class are located.

## 5.2. Supervised *t*-SNE with Triplet Loss

In this example, we combined a parametric version of *t*-SNE (see Section 4.2) with a triplet loss [CSSB10] to learn several supervised embeddings for the forest covertype dataset [BD99]. This is an example of an instance-level constraint as categorized by Vu et al. [VBF22].

The forest covertype dataset consists of 581,012 records with 54 attributes each. Each item corresponds to a $30\,\mathrm{m} \times 30\,\mathrm{m}$ cell of a US region, and the attributes describe cartographic variables, such as elevation, slope, and distance from the nearest roadway. Each item is labeled with the ground truth value for the type of trees covering the cell (e.g., aspen, krummholz, and spruce/fir). The dataset is strongly imbalanced, with the most prevalent class being more than 100 times more frequent than the least. In this example, we used the first ten numerical attributes and sampled an almost balanced subset of 7000 items.

Supervising *t*-SNE with an additional term based on triplets can be achieved easily thanks to the ParaDime interface: First, we use negative-edge sampling to construct triplets. In negative-edge sampling, rather than batches of individual items, batches of edges between items are sampled during training. In other applications of this sampling strategy (e.g., UMAP [MHM18]), a positive edge is sampled according to the probabilities of neighborhood of the two points (i.e., vertices). A specified number of random negative edges for one of the two vertices is then added. Negative edges are edges between two vertices for which the probability of neighborhood is zero. In this example, we instead created a probability matrix $r$ with $r_{ij} = 1$ if $g_i = g_j$ and 0 else, where $g_i$ are the ground truth

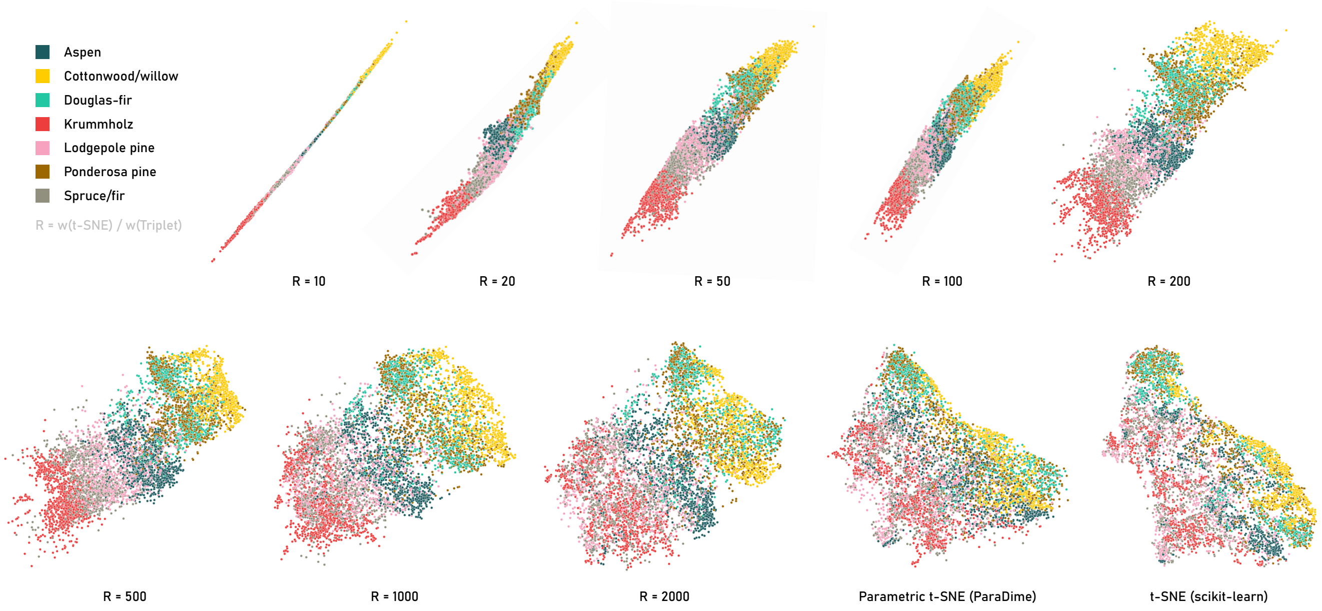

**Figure 4:** *Supervised embeddings of a subset of the forest covertype dataset [CSSB10]. All embeddings labeled with R are supervised versions of parametric t-SNE, where supervision was included by means of a triplet loss based on the ground truth labels. R is the ratio of the weights of the t-SNE loss and the triplet loss. For comparison, embeddings created with scikit-learn's non-parametric t-SNE implementation and with a plain ParaDime t-SNE version (using item-based sampling and no triplet loss) are shown. The perplexity was 200 in all cases, and a class-balanced subset of 7000 items was used.*

labels of the data. If we use this probability matrix for negative-edge sampling with a negative sampling rate of one, we essentially sample one pair of vertices $(a,b)$ with equal labels and another pair $(a,c)$ with different labels. The set of vertices $a,b,c$ constitutes a triplet [CSSB10; BRPM16]. We can then simply add an additional triplet loss. This results in the following ParaDime specification:

```
derived data: [<PCA>]
relations:
  - <global t-SNE rel>
  - <batch t-SNE rel>
  - name: r
    level: global
    type: pairwise eq
losses:
  - <PCA init loss>
  - <t-SNE loss>
  - name: triplet
    type: triplet
    func: margin
    keys:
```

```
      data: [main, data]
training phases:
  - <PCA init>
  - loss:
      components:
        - tsne
        - triplet
      weights: <w>
      sampling:
        type: edge
        options:
          rels: r
          rate: 1
```

Here, `pairwise eq` stands for the global relation as defined by $r_{ij}$, and `margin` is the name of the following loss function that is applied to the triplets [WSL*14; BRPM16]:

$$L_{\text{triplet}}(a,b,c) = \max(d(a,b) - d(a,c) + m, 0), \qquad (1)$$

where $m$ is the margin hyperparameter. We abridged the parts of the specification that match that of *t*-SNE from Section 4.2.

Figure 4 shows eight versions of embeddings specified this way, with different values for the loss weights. In all cases, the model was a fully connected neural network with hidden layer dimensions 100 and 50. Each embedding was initialized with a PCA-based pre-training for ten epochs with item sampling and a batch size of 500. As explained above, the main embedding phases used negative-edge sampling, with 300 triplets being sampled in each batch. For comparison, Figure 4 includes a parametric *t*-SNE without the extra triplet loss and with regular item sampling. We also show the result of scikit-learn's non-parametric *t*-SNE. For all embeddings the perplexity value was set to 200.

For the triplet loss as defined above to be minimal, the distance along negative edges (i.e., between a pair of items with different labels) must be substantially larger than the distances along a positive edge. This pulls together items from the same class. Putting too much weight onto the triplet loss causes all items to condense along a single line, approximately sorted by their class labels. As the weight of the triplet loss is reduced, the structure of the "pure" *t*-SNE is increasingly preserved, while classes are well separated (see, e.g., the embeddings for *t*-SNE/triplet loss weight ratios of 1000 in Figure 4). With vanishing weight on the triplet loss, the embedding still differs noticeably from that which used item-based sampling; here, the triplet sampling strategy might be disadvantageous, as it favors certain batch configurations over others.

One potential application idea for such supervised embeddings is an interactive visual interface for dataset exploration, that allows users to switch between a purely attribute-driven visualization (e.g., pure $t$-SNE) and a supervised one with more pronounced class separation. In the former, users could explore similarities and differences between all data points as usual, while the latter would enable class-specific exploration without losing track of the overall structure.

### 5.3. Attribute-guided Embeddings

In this example, we again look at embeddings of the covertype dataset discussed in the previous section. This time, however, our primary interest is not in the class distribution, but in using specific attributes to guide the embeddings. In particular, we used ParaDime to construct an embedding in which a specified direction correlates with one of the high-dimensional attributes. To this end, we defined a new type of loss:

$$L_{\text{corr}}(a,b;i,j) = 1 - \left(\frac{\operatorname{cov}(a_i,b_j)}{\sigma_{a_i}\sigma_{b_j}}\right)^2. \tag{2}$$

Here, $a$ and $b$ are two data matrices with the same number of rows, and $a_i$ and $b_j$ refer to columns $i$ and $j$, respectively; cov is the covariance, and $\sigma$ is the standard deviation. This loss is equivalent to one minus the squared Pearson's correlation coefficient for the $i$th column of $a$ and the $j$th column of $b$. During the training of our routine, $a$ will be a batch of high-dimensional data and $b$ the processed (i.e., embedded) 2-dimensional batch.

Having defined a loss `corr` that uses the function $L_{\text{corr}}$ (Eq. 2) and applies it to the unprocessed and embedded versions of the input batch, we can simply construct a compound loss analogously to the other examples in this section. The loss components ($t$-SNE loss and correlation loss) can be weighted, and the dimensions that should correlate can be specified as options to the loss.

Figure 5 shows four examples of such attribute-guided embeddings with different weights. In all examples, $i$ was set to eight and $j$ to one, which means that the *Hillshade (noon)* attribute of the covertype dataset was constrained to correlate with the $x$-direction of the embedding. In the embeddings in Figure 5, the points are colored by the high-dimensional attribute value specified. With increasing weight on the correlation loss, the embedding is distorted such that the values decrease from left to right, while the remaining structure is preserved to some extent. Within a certain range of weights, the transition from unguided to strongly guided embeddings appears to be smooth, with the points “folding over” continuously to satisfy the constraints.

Because ParaDime models are neural networks, we can apply to them any existing explanation technique developed for neural networks. In this example, we sought to verify that the attribute we specified (feature eight, *Hillshade (noon)*) was actually of high importance for the resulting $x$ value. To this end, we applied a “vanilla” version of integrated gradients [Mol22] to our model. The resulting feature importance scores are shown in the bar chart in Figure 5. Note that for the strongly guided embedding, feature eight is indeed the most important for the $x$ result by some margin, and it does not contribute to $y$ at all.

Attribute-guided embeddings are not only a showcase for how easily new techniques can be constructed with ParaDime. They might be useful in cases in which users want to transition from purely unsupervised embeddings to ones where a specified attribute is of particular interest to the analysis.

## 6. Discussion

In this section, we discuss some of the design choices related to the structure of the ParaDime grammar and its implementation. We also reflect on ParaDime's ease of use, its customizability, limitations, and future work.

### 6.1. Structure of the Grammer

The structure of the ParaDime grammar cannot be uniquely derived from the necessary building blocks (dataset, relations, etc.), but depends on a number of choices. For example, in an earlier version of the specifications, losses were defined entirely within the training phases, and their specification included a weight. However, this strongly limited the reusabilty of losses across phases. We thus opted for loss specifications at the base level, which required the introduction of the `components` and `weights` entries, and the use of loss names that could be referenced.

Furthermore, we initially planned different base-level entries for lists of global and batch-wise relations. From a computational view, they are typically used at different times in the routines, and only the batch-wise relations must be differentiable. Nevertheless, we ultimately chose a flat list of relations with individual `level` entries to highlight the conceptual similarities between them.

Initially, we had also planned to include a `model` entry in the ParaDime specifications. Our first draft included a nested structure of (sub-)model specifications based closely on how PyTorch allows arbitrarily nested modules. However, we soon realized that the creation of a general declarative grammar for neural networks went well beyond the scope of this work. We thus decided to have users pass their PyTorch module to ParaDime alongside a DR specification. ParaDime can also construct a default, fully connected model to help users to get started.

### 6.2. Implementation Choices

We considered PyTorch [PGM*19], TensorFlow [AAB*16], and JAX [FJL18] as machine learning frameworks for ParaDime. Ultimately, we settled on PyTorch because it has become the most popular framework for research purposes [OCo21].

While in this work we used YAML [dNMP*21] for the specifications in this paper due to its focus on readability, ParaDime is also capable of parsing JSON specifications with the same structures. In addition to construction by specifications (which facilitate sharing and reproducibility) ParaDime allows an object-oriented construction of routines, as this is particularly suitable for adapting existing routines or dynamically changing properties of routines.

### 6.3. Ease of Use and Customization

As outlined in Section 4.4, ParaDime can be readily used for distance- and neighbor-based DR techniques. We asked an AI Bachelor student with no prior deep learning experience to implement a parametric version of Isomap using ParaDime. Without

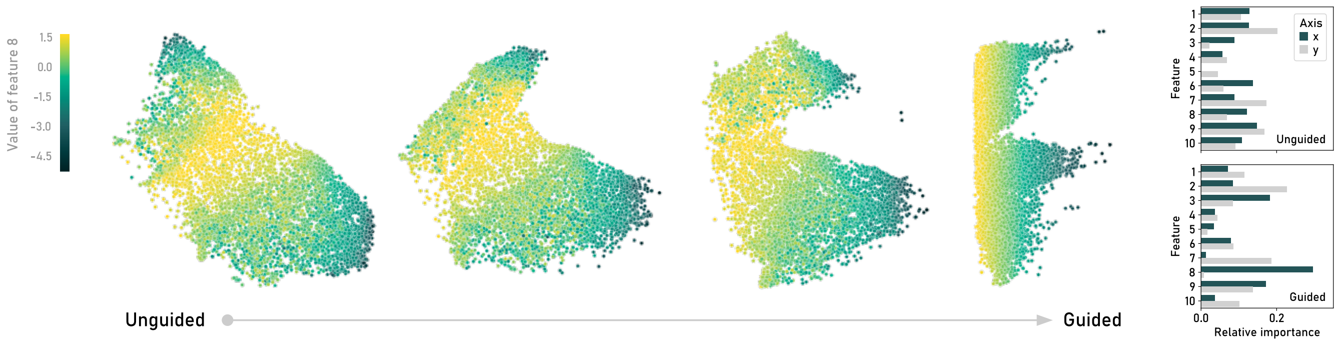

**Figure 5:** *Attribute-guided embeddings of a subset of the forest covertype dataset [CSSB10]. Attribute guiding was implemented by combining t-SNE with a correlation loss which orders the data points along the x-axis by the value of the eighth feature (hillshade at noon). The weights for the embeddings shown are* $(w_{t\text{-}SNE}, w_{corr}) = (1,0)$, $(5000,1)$, $(1000,1)$, *and* $(100,1)$, *respectively. The bar chart on the right shows, based on integrated gradients, the feature importance scores for the learned embeddings.*

ParaDime, the student would have had to write a sampling routine that correctly incorporates pairwise relations, set up the PyTorch module, write and correctly apply the embedding loss, and set up the optimization loops. With ParaDime, the student only had to wrap code for the geodesic distance computation (taken from scikit-learn) in a ParaDime relations class. Because ParaDime already offers differentiable implementations of several batch-wise relations, the student did not have to learn PyTorch at all.

For more obscure DR techniques, users must program custom losses or batch-wise relations. Ensuring that all relevant parts remain differentiable requires some understanding of PyTorch. Currently, the sampling procedure is the most difficult part of the routines to customize, since it is not directly accessible through the ParaDime API. However, we believe that the built-in item- and edge-based samplers should suffice for most cases. Even in highly customized applications, ParaDime should reduce overhead because it takes care of most of the data handling, facilitates the combination of multiple losses, and/or sets up the training loops.

### 6.4. Limitations & Future Work

One major limitation when moving from traditional DR techniques to parametric embeddings is the increased number of hyperparameters. Users must select a suitable model architecture and set batch sizes, optimizers and learning rates such that the loss is properly minimized. For the predefined ParaDime routines, we provide defaults based on our own experiments. With new routines, however, finding suitable choices for hyperparameters can be challenging. The same is true for weights in compound losses. Choosing suitable weights for the loss components is a long-standing problem in multitask learning [GLS*19]. As a result, non-obvious weight ratios have to be tried out, as seen in some of the examples discussed in Section 5. However, ParaDime's focus on reusability and ease of specification facilitates experiments with different weights. ParaDime also features built-in plotting utilities, which allow users to rapidly check the embeddings visually.

Another limitation related to batch-wise training is that certain global constraints are difficult to implement. For example, global density-based measures such as that used in densMAP [NBC21] are challenging to reproduce from small batches. In principle, the batch size in ParaDime can be set to the number of items in the dataset to allow computation of global measures during training. However, this might lead to problems with gradients for other losses. We plan to experiment with such globally constrained techniques to provide better ways of incorporating them.

Finally, we plan to include export utilities for the trained models so that they can easily be used elsewhere. It would be particularly desirable to export models in a format that could be used directly within a web-browser. Visualizations implemented as web-apps could thus make use of pre-trained ParaDime routines without the need for a backend.

## 7. Conclusion

We have introduced ParaDime, a framework for parametric dimensionality reduction. The ParaDime grammar allows users to specify DR routines in a declarative way. We have shown how this approach enables parametric extension of existing techniques and illustrated how ParaDime facilitates experimentation with new ideas. We hope that—due to our focus on flexibility and customization—ParaDime will inspire further research into the potential of parametric dimensionality reduction.

## Acknowledgements

This work was supported by the State of Upper Austria and the Austrian Federal Ministry of Education, Science and Research via the Linz Institute of Technology (LIT-2019-7-SEE-117), the State of Upper Austria (Human-Interpretable Machine Learning), the Austrian Science Fund (FWF DFH 23–N), and the Austrian Research Promotion Agency (FFG 881844). Pro$^2$Future is funded within the Austrian COMET Program under the auspices of the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology, the Austrian Federal Ministry for Digital and Economic Affairs, and of the States of Upper Austria and Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

## References

[AAB*16] Abadi, M., Agarwal, A., Barham, P., et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". *arXiv:1603.04467 [cs]* (Mar. 14, 2016). DOI: 10.48550/arXiv.1603.04467. arXiv: 1603.04467 9.

[AAB21] Agrawal, A., Ali, A., and Boyd, S. "Minimum-Distortion Embedding". *Foundations and Trends in Machine Learning* 14.3 (2021), 211–378. DOI: 10.1561/2200000090 2.

[AW10] Abdi, H. and Williams, L. J. "Principal component analysis". *Wiley Interdisciplinary Reviews: Computational Statistics* 2.4 (2010), 433–459. DOI: 10.1002/wics.101 1, 2.

[BBK22] Böhm, J. N., Berens, P., and Kobak, D. "Attraction-Repulsion Spectrum in Neighbor Embeddings". *Journal of Machine Learning Research* 23.95 (2022), 1–32. URL: http://jmlr.org/papers/v23/21-0055.html 2.

[BCV13] Bengio, Y., Courville, A., and Vincent, P. "Representation Learning: A Review and New Perspectives". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), 1798–1828. DOI: 10.1109/TPAMI.2013.50 2.

[BD99] Blackard, J. A. and Dean, D. J. "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables". *Computers and Electronics in Agriculture* 24.3 (Dec. 1999), 131–151. DOI: 10.1016/S0168-1699(99)00046-0 7.

[BDR*04] Bengio, Y., Delalleau, O., Roux, N. L., et al. "Learning Eigenfunctions Links Spectral Embedding and Kernel PCA". *Neural Computation* 16.10 (Oct. 1, 2004), 2197–2219. DOI: 10.1162/0899766041732396 2.

[BMH*19] Becht, E., McInnes, L., Healy, J., et al. "Dimensionality reduction for visualizing single-cell data using UMAP". *Nature Biotechnology* 37.1 (Jan. 2019), 38–44. DOI: 10.1038/nbt.4314 2.

[Bot10] Bottou, L. "Large-Scale Machine Learning with Stochastic Gradient Descent". *Proceedings of COMPSTAT'2010*. Ed. by Lechevallier, Y. and Saporta, G. Heidelberg: Physica-Verlag HD, 2010, 177–186. ISBN: 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3_16 4.

[BRPM16] Balntas, V., Riba, E., Ponsa, D., and Mikolajczyk, K. "Learning local feature descriptors with triplets and shallow convolutional neural networks". *Procedings of the British Machine Vision Conference 2016*. York, UK: British Machine Vision Association, 2016, 119.1–119.11. ISBN: 978-1-901725-59-9. DOI: 10.5244/C.30.119 8.

[Car97] Caruana, R. "Multitask Learning". *Machine Learning* 28.1 (1997), 41–75. DOI: 10.1023/A:1007379606734 7.

[CC08] Cox, M. A. A. and Cox, T. F. "Multidimensional Scaling". Chen, C.-h., Härdle, W., and Unwin, A. *Handbook of Data Visualization*. Berlin, Heidelberg: Springer, 2008, 315–347. ISBN: 978-3-540-33037-0. DOI: 10.1007/978-3-540-33037-0_14 2.

[CG15] Cunningham, J. P. and Ghahramani, Z. "Linear Dimensionality Reduction: Survey, Insights, and Generalizations". *Journal of Machine Learning Research* 16.1 (2015), 2859–2900 2, 4.

[CSSB10] Chechik, G., Sharma, V., Shalit, U., and Bengio, S. "Large Scale Online Learning of Image Similarity Through Ranking". *Journal of Machine Learning Research* 11.36 (2010), 1109–1135. URL: https://www.jmlr.org/papers/v11/chechik10a.html 4, 7, 8, 10.

[DBHK22] Damrich, S., Böhm, J. N., Hamprecht, F. A., and Kobak, D. "From *t*-SNE to UMAP with contrastive learning". *arXiv:2206.01816 [cs.LG]* (2022). DOI: 10.48550/arXiv.2206.01816 2.

[dBMVL19] De Bodt, C., Mulders, D., Verleysen, M., and Lee, J. A. "Nonlinear Dimensionality Reduction With Missing Data Using Parametric Multiple Imputations". *IEEE Transactions on Neural Networks and Learning Systems* 30.4 (Apr. 2019), 1166–1179. DOI: 10.1109/TNNLS.2018.2861891 2.

[DH21] Damrich, S. and Hamprecht, F. A. "On UMAP's True Loss Function". *Advances in Neural Information Processing Systems* 34 (2021), 5798–5809. URL: https://proceedings.neurips.cc/paper/2021/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html 2.

[dNMP*21] Döt Net, I., Müller, T., Pantelis, A., et al. *YAML Ain't Markup Language (YAML™) version 1.2*. Revision 1.2.2. YAML Language Development Team, Oct. 1, 2021. URL: https://yaml.org/spec/1.2.2 3, 9.

[EHJT04] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. "Least angle regression". *The Annals of Statistics* 32.2 (Apr. 1, 2004). DOI: 10.1214/009053604000000067 4, 5.

[EMK*19] Espadoto, M., Martins, R. M., Kerren, A., et al. "Toward a Quantitative Survey of Dimension Reduction Techniques". *IEEE Transactions on Visualization and Computer Graphics* 27 (3 2019), 2153–2173. DOI: 10.1109/TVCG.2019.2944182 4, 5, 7.

[FJL18] Frostig, R., Johnson, M. J., and Leary, C. "Compiling machine learning programs via high-level tracing". *SysML Conference*. 2018, 3. URL: https://mlsys.org/Conferences/2019/doc/2018/146.pdf 9.

[GC20] Graving, J. M. and Couzin, I. D. "VAE-SNE: a deep generative model for simultaneous dimensionality reduction and clustering". *bioRxiv* (2020). DOI: 10.1101/2020.07.17.207993 6.

[GHM*22] Görtler, J., Hohman, F., Moritz, D., et al. "Neo: Generalizing Confusion Matrix Visualization to Hierarchical and Multi-Output Labels". *CHI Conference on Human Factors in Computing Systems*. New Orleans LA USA: ACM, Apr. 29, 2022, 1–13. ISBN: 978-1-4503-9157-3. DOI: 10.1145/3491102.3501823 3.

[GLS*19] Gong, T., Lee, T., Stephenson, C., et al. "A Comparison of Loss Weighting Strategies for Multi task Learning in Deep Neural Networks". *IEEE Access* 7 (2019), 141627–141632. DOI: 10.1109/ACCESS.2019.2943604 10.

[GMH12] Gisbrecht, A., Mokbel, B., and Hammer, B. "Linear basis-function t-SNE for fast nonlinear dimensionality reduction". *International Joint Conference on Neural Networks (IJCNN 2012)*. Brisbane, Australia: IEEE, June 2012, 1–8. DOI: 10.1109/IJCNN.2012.6252809 2.

[GSH15] Gisbrecht, A., Schulz, A., and Hammer, B. "Parametric nonlinear dimensionality reduction using kernel t-SNE". *Neurocomputing* 147 (Jan. 2015), 71–82. DOI: 10.1016/j.neucom.2013.11.045 2.

[Hin06] Hinton, G. E. "Reducing the Dimensionality of Data with Neural Networks". *Science* 313.5786 (July 28, 2006), 504–507. DOI: 10.1126/science.1127647 2.

[Hin23a] Hinterreiter, A. *ParaDime: A Framework for Parametric Dimensionality Reduction*. Documentation. 2023. URL: https://paradime.readthedocs.io/en/latest/ 4.

[Hin23b] Hinterreiter, A. *ParaDime: A Framework for Parametric Dimensionality Reduction*. GitHub repository. 2023. URL: https://github.com/jku-vds-lab/paradime 4.

[HR02] Hinton, G. E. and Roweis, S. T. "Stochastic Neighbor Embedding". *Advances in Neural Information Processing Systems* 15 (2002), 8. URL: https://proceedings.neurips.cc/paper/2002/hash/6150ccc6069bea6b5716254057a194ef-Abstract.html 2.

[HZ93] Hinton, G. E. and Zemel, R. S. "Autoencoders, minimum description length and Helmholtz free energy". *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS '93)*. 1993, 3–10. DOI: 10.5555/2987189.2987190 2.

[KB17] Kingma, D. P. and Ba, J. "Adam: A Method for Stochastic Optimization". *arXiv:1412.6980 [cs]* (Jan. 29, 2017). DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 4, 5.

[KL21] Kobak, D. and Linderman, G. C. "Initialization is critical for preserving global data structure in both t-SNE and UMAP". *Nature Biotechnology* 39.2 (Feb. 2021), 156–157. DOI: 10.1038/s41587-020-00809-z 2.

[Kru64] KRUSKAL, J. B. "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". *Psychometrika* 29.1 (Mar. 1964), 1–27. DOI: 10.1007/BF02289565 5.

[Kul12] KULIS, B. "Metric learning: A survey". *Foundations and Trends in Machine Learning* 5.4 (2012), 287–364. DOI: 10.1561/2200000019 2.

[LeC05] LECUN, Y. *The MNIST database of handwritten digits*. 2005. URL: http://yann.lecun.com/exdb/mnist/ (visited on 09/10/2022) 1, 5–7.

[LKL*22] LAI, C.-H., KUO, M.-F., LIEN, Y.-H., et al. "Parametric Dimension Reduction by Preserving Local Structure". *2022 IEEE Visualization Conference – Short Papers*. 2022, 75–79. DOI: 10.1109/VIS54862.2022.00024 2, 5.

[LWLG22] L'YI, S., WANG, Q., LEKSCHAS, F., and GEHLENBORG, N. "Gosling: A Grammar-based Toolkit for Scalable and Interactive Genomics Data Visualization". *IEEE Transactions on Visualization and Computer Graphics* 28.1 (Jan. 2022), 140–150. DOI: 10.1109/TVCG.2021.3114876 3.

[MHM18] MCINNES, L., HEALY, J., and MELVILLE, J. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". *arXiv:1802.03426 [cs, stat]* (2018). DOI: 10.48550/arXiv.1802.03426 1, 2, 4, 7.

[Mol22] MOLNAR, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: christophm.github.io/interpretable-ml-book/ 9.

[MSC*13] MIKOLOV, T., SUTSKEVER, I., CHEN, K., et al. "Distributed Representations of Words and Phrases and their Compositionality". *Advances in Neural Information Processing Systems*. Vol. 26. 2013. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html 4.

[MvdMY*10] MIN, M. R., van der MAATEN, L., YUAN, Z., et al. "Deep Supervised t-Distributed Embedding". *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010. URL: https://icml.cc/Conferences/2010/papers/149.pdf 2.

[NBC21] NARAYAN, A., BERGER, B., and CHO, H. "Assessing single-cell transcriptomic variability through density-preserving data visualization". *Nature Biotechnology* 39.6 (June 2021), 765–774. DOI: 10.1038/s41587-020-00801-7 10.

[OCo21] O'CONNOR, R. *PyTorch vs TensorFlow in 2022*. AssemblyAI Blog. 2021. URL: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/ (visited on 09/29/2022) 9.

[PDFE18] PARK, D., DRUCKER, S. M., FERNANDEZ, R., and ELMQVIST, N. "Atom: A Grammar for Unit Visualizations". *IEEE Transactions on Visualization and Computer Graphics* 24.12 (2018), 3032–3043. DOI: 10.1109/TVCG.2017.2785807 3.

[Pea01] PEARSON, K. "On lines and planes of closest fit to systems of points in space". *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (Nov. 1901), 559–572. DOI: 10.1080/14786440109462720 2.

[PGM*19] PASZKE, A., GROSS, S., MASSA, F., et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems* 32 (2019), 12. URL: https://papers.nips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html 4, 9.

[PSZ19] POLIČAR, P. G., STRAŽAR, M., and ZUPAN, B. "openTSNE: a modular Python library for t-SNE dimensionality reduction and embedding". *bioRxiv* (2019). DOI: 10.1101/731877 5.

[PVG*11] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), 2825–2830. URL: https://dl.acm.org/doi/10.5555/1953048.2078195 5.

[SMDH13] SUTSKEVER, I., MARTENS, J., DAHL, G., and HINTON, G. "On the importance of initialization and momentum in deep learning". *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. 2013, 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html 4.

[SMG21] SAINBURG, T., MCINNES, L., and GENTNER, T. Q. "Parametric UMAP Embeddings for Representation and Semisupervised Learning". *Neural Computation* 33.11 (Oct. 12, 2021), 2881–2907. DOI: 10.1162/neco_a_01434 2, 6.

[SMWH17] SATYANARAYAN, A., MORITZ, D., WONGSUPHASAWAT, K., and HEER, J. "Vega-Lite: A Grammar of Interactive Graphics". *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), 341–350. DOI: 10.1109/TVCG.2016.2599030 3.

[SRHH16] SATYANARAYAN, A., RUSSELL, R., HOFFSWELL, J., and HEER, J. "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization". *IEEE Transactions on Visualization and Computer Graphics* 22.1 (Jan. 31, 2016), 659–668. DOI: 10.1109/TVCG.2015.2467091 3.

[Ten00] TENENBAUM, J. B. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". *Science* 290.5500 (Dec. 22, 2000), 2319–2323. DOI: 10.1126/science.290.5500.2319 2, 6.

[TLZM16] TANG, J., LIU, J., ZHANG, M., and MEI, Q. "Visualizing Large-scale and High-dimensional Data". *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Geneva: International World Wide Web Conferences Steering Committee, Apr. 11, 2016, 287–297. ISBN: 978-1-4503-4143-1. DOI: 10.1145/2872427.2883041 4, 6.

[Tor52] TORGERSON, W. S. "Multidimensional Scaling: I. Theory and Method". *Psychometrika* 17.4 (Dec. 1952), 401–419. DOI: 10.1007/BF02288916. URL: http://www.galileoco.com/literature/Torgerson52.pdf 2.

[VBF22] VU, V. M., BIBAL, A., and FRENAY, B. "Integrating Constraints into Dimensionality Reduction for Visualization: a Survey". *IEEE Transactions on Artificial Intelligence* (2022), 1–19. DOI: 10.1109/TAI.2022.3204734 2, 7.

[vdMaa09] Van der MAATEN, L. "Learning a Parametric Embedding by Preserving Local Structure". *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Vol. 5. Proceedings of Machine Learning Research. 2009, 384–391. URL: http://proceedings.mlr.press/v5/maaten09a.html 2.

[vdMH08] Van der MAATEN, L. and HINTON, G. "Visualizing Data using t-SNE". *Journal of Machine Learning Research* 9 (2008), 2579–2605. URL: https://jmlr.org/papers/v9/vandermaaten08a.html 1–5.

[VK01] VENNA, J. and KASKI, S. "Neighborhood Preservation in Nonlinear Projection Methods: An Experimental Study". *Artificial Neural Networks — ICANN 2001*. Ed. by DORFFNER, G., BISCHOF, H., and HORNIK, K. Red. by GOOS, G., HARTMANIS, J., and van LEEUWEN, J. Vol. 2130. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, 485–491. ISBN: 978-3-540-42486-4. DOI: 10.1007/3-540-44668-0_68 7.

[Wil02] WILLIAMS, C. K. "On a Connection between Kernel PCA and Metric Multidimensional Scaling". *Machine Learning* 46 (2002), 11–19. DOI: 10.1023/A:1012485807823 2.

[Won20] WONGSUPHASAWAT, K. "Encodable: Configurable Grammar for Visualization Components". *IEEE Visualization Conference (VIS'20)*. IEEE Visualization Conference (VIS'20). IEEE, 2020, 131–135. ISBN: 978-1-72818-014-4. DOI: 10.1109/VIS47514.2020.00033 3.

[WSL*14] WANG, J., SONG, Y., LEUNG, T., et al. "Learning Fine-Grained Image Similarity with Deep Ranking". *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH, USA: IEEE, June 2014, 1386–1393. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.180 8.