The DipEncoder: Enforcing Multimodality in Autoencoders

Collin Leiber
LMU Munich
Munich, Germany
leiber@dbs.ifi.lmu.de

Lena G. M. Bauer
Faculty of Computer Science
UniVie Doctoral School Computer Science
ds:UniVie
University of Vienna, Vienna, Austria
lena.bauer@univie.ac.at

Michael Neumayr
LMU Munich
Munich, Germany
michael.neumayr@campus.lmu.de

Claudia Plant
Faculty of Computer Science
ds:UniVie
University of Vienna, Vienna, Austria
claudia.plant@univie.ac.at

Christian Böhm
Faculty of Computer Science
University of Vienna, Vienna, Austria
christian.boehm@univie.ac.at

ABSTRACT

Hartigan's Dip-test of unimodality gained increasing interest in unsupervised learning over the past few years. It is free from complex parameterization and does not require a distribution assumed a priori. A useful property is that the resulting Dip-values can be derived to find a projection axis that identifies multimodal structures in the data set. In this paper, we show how to apply the gradient not only with respect to the projection axis but also with respect to the data to improve the cluster structure. By tightly coupling the Dip-test with an autoencoder, we obtain an embedding that clearly separates all clusters in the data set. This method, called DipEncoder, is the basis of a novel deep clustering algorithm. Extensive experiments show that the DipEncoder is highly competitive to state-of-the-art methods.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Computing methodologies** → **Cluster analysis**; *Dimensionality reduction and manifold learning*; *Neural networks.*

KEYWORDS

Hartigan's Dip-test, Deep Clustering, Dimensionality reduction

ACM Reference Format:
Collin Leiber, Lena G. M. Bauer, Michael Neumayr, Claudia Plant, and Christian Böhm. 2022. The DipEncoder: Enforcing Multimodality in Autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539407

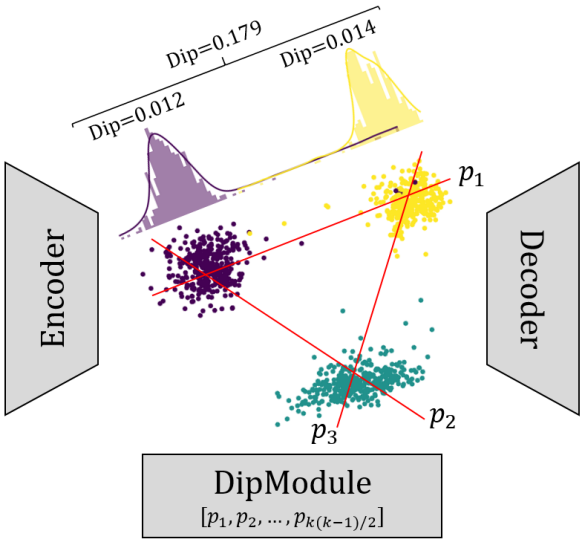


Figure 1: Architecture of the DipEncoder. The illustration shows the two-dimensional result of the DipEncoder on a subset of the Optdigits data set. We can see that each combination of clusters receives its own projection axis within the DipModule. The histogram depicts the purple and yellow clusters projected onto their corresponding projection axis p_1. In addition, the figure shows the Dip-values of the purple cluster (0.012), the yellow cluster (0.014), and the combination of the two (0.179). From this, we can conclude that both clusters are unimodal while the combination is multimodal.

1 INTRODUCTION

The interest in analyzing large amounts of high-dimensional data such as images, videos or texts increased significantly in recent years. Since such data sets are highly complex both in terms of their interpretability and the time it takes to process them, particular analysis methods are usually required.

An established strategy to handle high-dimensional data is to run a dimensionality reduction technique before the desired analysis method. The most common technique is probably the Principal Component Analysis (PCA) [10]. With the growing availability of computing power, Neural Networks (NNs) have also become more popular. Due to their high abstraction capabilities, they offer a range of powerful analysis options. Special architectures can even favor the performance concerning certain data types (e.g.,

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539407

Convolutional Neural Networks (CNNs) [23] for image data). A strategy that combines dimensionality reduction with NNs is to make use of an autoencoder (AE) [1]. This feed-forward NN learns a lower-dimensional embedding of the input data set. One can then execute further analysis procedures on this embedding.

In recent years, the field of clustering has increasingly taken up this idea. Corresponding procedures, also referred to as Deep Clustering (DC) methods, usually use an AE to learn an embedding in which the actual clustering procedure is applied, allowing the clustering objective to be updated simultaneously to the embedding. Hereby, the runtime can drop significantly due to the lower dimensionality and we counteract the curse of dimensionality while at the same time using the abstraction capabilities of the AE.

In this paper, we present the DipEncoder. This NN combines an AE with Hartigan's Dip-test of unimodality [14]. The Dip-test is a parameter-free statistical test that returns a Dip-value within the interval $(0, 0.25]$, which specifies the multimodality of a one-dimensional data set. Dip-values close to 0 indicate unimodality of the input samples, while larger values indicate that the samples contain at least two modes. Our goal is to use this test to create an embedding that clearly separates different groups of data. In other terms, we want to achieve an embedding that shows high modality between each combination of clusters. Figure 1 illustrates this idea by the high Dip-value of 0.179 between the purple and yellow cluster. However, we cannot simply maximize the Dip-value since we have to be careful that this process does not pull one of the clusters apart to achieve a multimodal structure. Therefore, we also want to minimize the modality within the separate clusters. In Figure 1 this corresponds to the small Dip-values of 0.012 for the purple and 0.014 for the yellow cluster. Since the Dip-test can only process one-dimensional samples, we create individual projection axes for each combination of clusters and store them within the so-called DipModule. These axes are represented in Figure 1, for example, by the red line $p1$ between the purple and yellow cluster. Using this architecture, we can leverage the gradient of the Dip-test to optimize the projection axes simultaneously to the embedding. We further use this idea to show how we can use the Dip-test to update the cluster labels. Building on this, we present a novel DC algorithm based solely on the Dip-test which does not require clustering-specific parameters other than the number of clusters k.

Our main contributions can be summarized as follows:

- First, we present the previously unused gradient of the Dip-value with respect to the data.
- We show how to use the gradients of the Dip-value in combination with an AE for supervised dimensionality reduction.
- Based on this, we develop a procedure that updates the cluster labels using only the Dip-test.
- This method is extended to create a novel Deep Clustering algorithm that does not require clustering-specific parameters apart from the number of clusters.

2 RELATED WORK

In the following, we describe a few methodologies that underlie our proposal. First, we describe the Dip-test in detail as it is an essential part of our procedure. Then, we briefly discuss dimensionality reduction techniques and basic Deep Clustering algorithms.

The Dip-test: The Dip-test [14] of unimodality is a statistical test that measures how multimodal a given one-dimensional data set is. It returns a Dip-value $\text{dip} \in (0, 0.25]$, which indicates unimodality if it is close to zero. Since the Dip-test is parameter-free, we do not have to state any underlying distribution function. Therefore, $\text{dip} \approx 0$ regardless of whether we execute the Dip-test on samples from a single Gaussian, Laplacian, uniform, or any other unimodal distribution. On the other hand, $0 \ll \text{dip} \leq 0.25$ if we have samples from distributions with multiple distinctive modes. Hartigan and Hartigan showed how to efficiently calculate the Dip-value on a sorted data set of size N with a complexity of $O(N)$ [14]. For this purpose, a modal interval is used, which indicates the steepest slope in the empirical cumulative distribution function (ECDF). These characteristics make the Dip-test very interesting for the Data Science community, which is why it has already been used for various purposes.

Since no distribution of the data has to be assumed, it is particularly suitable for unsupervised learning. One of the first clustering algorithms using the Dip-test is Dip-means [19] which aims at identifying the number of clusters. Here, the Dip-test is calculated on the distances between all objects within a cluster. A new cluster is added if the distances do not show a unimodal distribution. Projected Dip-means [2] pursues the same objective but projects the data onto projection axes and applies the Dip-test on these projected one-dimensional values. SkinnyDip [28] recursively analyzes the features of a data set by interpreting each modal interval returned by the Dip-test as a cluster. This idea is continued by StrDip [27] to cluster streaming data. Nr-Dipmeans [30] attempts to determine the number of clusters in a non-redundant clustering setting. Therefore, each cluster is split into two, and the objects are projected onto the line connecting the two new centers. The initial cluster is kept if the Dip-test indicates unimodality; otherwise, the procedure continues with the two new clusters. The first clustering algorithm that combines the Dip-test with Deep Learning is DipDECK [25]. Here, clustering is done in the embedded space of an AE. Initially, the technique heavily overestimates the number of clusters. If the data points within two microclusters show high unimodality, they are merged, and the embedding can be further updated.

DipTransformation [34] shows that the Dip-test can also be used for data preprocessing. It uses the Dip-test to transform and scale a data set so that its most important features are highlighted. [28] and [33] go one step further. They take advantage of the fact that we can deduce the gradient of the Dip-value with respect to the projection axis, as shown in [22]. This allows them to identify cluster-friendly subspaces by picking out those projections which yield the highest multimodality. The differentiability is a convenient feature of the Dip-test. We want to extend this idea by additionally using the gradient with respect to the data.

Dimensionality reduction techniques: Methods that reduce the dimensionality of a data set fall into two main categories: unsupervised and supervised.

Principal Component Analysis (PCA) [10] is probably the best-known dimensionality reduction technique and is a common preprocessing step when analyzing a data set. It rotates the features space such that only the components with the highest variances remain. Independent Component Analysis (ICA) [18] is not a dimensionality reduction technique per definition. However, it also

finds a lower-dimensional basis of the data by searching for statistically independent components. t-Distributed Stochastic Neighbor Embedding (t-SNE) [35] tries to preserve local structures in high-dimensional data sets while reducing the dimensionality. Therefore, it converts distances in the original feature space into probabilities of whether an object would pick another as a neighbor. Uniform Manifold Approximation and Projection (UMAP) [31] pursues a similar goal. However, it uses a more extensive mathematical basis by leveraging Riemannian geometry. Another way to perform a dimensional reduction is to use an autoencoder (AE) [1]. This unsupervised NN consists of two parts, an encoder enc(·) that transforms the input to the embedding and a decoder dec(·) that tries to restore the data from the embedding to its original state. This basic idea is also presented in Figure 1, where we have a two-dimensional embedding. The AE is usually trained by a batch-wise optimization of the reconstruction loss $\mathcal{L}_{rec}$.

$$\mathcal{L}_{rec}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} ||x - \text{dec}(\text{enc}(x))||_2^2, \tag{1}$$

where $\mathcal{B} \subseteq X$ is one batch of the data set $X \subseteq \mathbb{R}^d$ and $||\cdot||_2^2$ denotes the squared Euclidean distance.

Until now, we only discussed unsupervised techniques. However, some methods use known cluster labels to achieve a lower dimensionality. One such supervised approach is the Linear Discriminant Analysis (LDA) [9]. LDA identifies a subspace by minimizing the intra-cluster variance while maximizing the inter-cluster variance. Another approach is Partial Least Squares (PLS) [36]. This method searches for structures in the data set that maximize the covariance with the labels.

We present the DipEncoder, which also falls into the group of supervised dimensionality reduction techniques since we want to improve the embedding of an AE by arranging objects of a common group unimodally and those of different groups multimodally.

Many of the mentioned methods have already been successfully combined with data-mining approaches to, for instance, create subspace or non-redundant clustering algorithms. Examples are Orth [5] (using PCA), LDA-k-means [6] (using LDA) or generally DC algorithms (using AEs).

Deep Clustering: Since our final product is a Deep Clustering (DC) algorithm, we want to briefly discuss corresponding procedures. One of the first methods that combine a simple AE with a centroid-based clustering objective is DEC [38] and its successor IDEC [13]. They optimize their network by minimizing the Kullback-Leibler divergence between soft cluster labels and an auxiliary target distribution. While DEC uses the reconstruction loss of the AE only for pretraining, IDEC integrates it into the cost function of the primary clustering method. DCN [39] is more oriented towards the original k-means algorithm and therefore uses hard cluster labels. The hierarchical clustering algorithm DeepECT [29] provides multiple levels of labels, which can then be analyzed at a later stage. Other methods introduce specific AE architectures to better handle certain types of data. For example VaDE [17] uses a Variational Autoencoder (VAE) [21] to solve a probabilistic clustering objective. DEKM [12], JULE [40] and DEPICT [7] all use CNNs, which greatly increases their processing power on image data. ClusterGAN [32] applies yet another clustering strategy by employing Generative Adversarial Networks (GANs) [11].

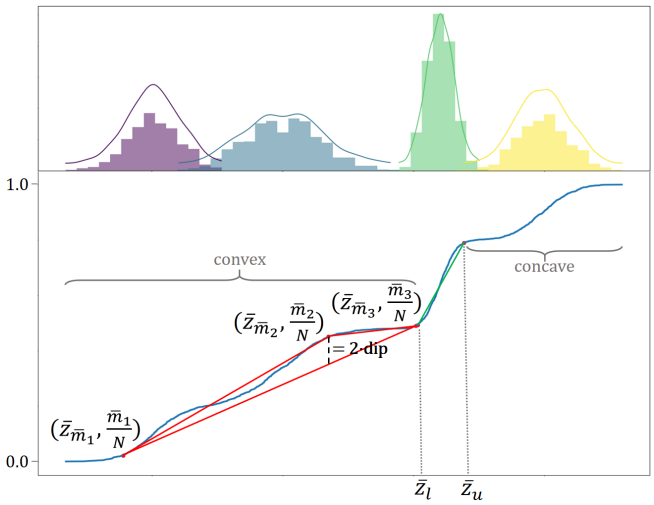


Figure 2: The figure shows the calculation of the Dip-value using an exemplary data set with four modes. [Top] Histogram of the data set. The colors indicate the underlying creation process. [Bottom] The blue line shows the ECDF of the samples. Between $\bar{z}_l$ and $\bar{z}_u$, the green line indicates the region of the maximum slope within the ECDF and thus the main mode(s) of the data set. There must be a convex distribution to the left of $\bar{z}_l$ and a concave distribution to the right of $\bar{z}_u$. The vertical height of the modal triangle (in red), formed by $(\bar{z}_{\bar{m}_1}, \frac{\bar{m}_1}{N})$, $(\bar{z}_{\bar{m}_2}, \frac{\bar{m}_2}{N})$ and $(\bar{z}_{\bar{m}_3}, \frac{\bar{m}_3}{N})$, shows the maximum deviation between the ECDF and a convex distribution and ultimately specifies the Dip-value.

We want to avoid such architecture-based extensions to show that our good results are based only on our core idea. However, extensions like convolutional layers could still be added to later evolutions of our technique.

3 THE DIPENCODER

The DipEncoder is an extension of an ordinary AE, which is able to process the gradients of the Dip-value. Therefore, we would like to discuss the mathematical foundation of those gradients first. An overview of the used symbols can be found in appendix A.

Typically, the objects within a data set are fixed, which is why so far, only the gradient with respect to the projection axis was used (e.g., in [22, 28, 33]). Since we are working within the embedding of an AE, we can also process the gradient regarding the data. To the best of our knowledge, we are the first who use the gradient of the Dip-value in a Deep Learning environment and generally the first who use the gradient with respect to the data.

For the computation of the Dip-value, we need to transform the embedded data set $Z = \text{enc}(X)$ into a one-dimensional space. Therefore, we assign distinct projection axes to each pair of clusters. These axes are stored in a separate shallow NN we call DipModule. It consists of $k(k-1)/2 \times m$ neurons, where k is the number of clusters and m the dimensionality of the embedding. We can now project the embedded samples of clusters a, denoted by $X_a \subseteq X$, and b, denoted by $X_b \subseteq X$, onto their corresponding axis $p_{a,b}$. We

define $X_{a,b} = X_a \cup X_b$ as the set of all objects assigned to either cluster a or cluster b and consequently $Z_{a,b} = \text{enc}(X_a \cup X_b)$. Then the projection is performed as $p_{a,b}^T \cdot z$, where $z \in Z_{a,b}$. Afterwards, we sort our projected embedded data to receive $\overline{Z}_{a,b} \subseteq \mathbb{R}^1$.

To calculate the Dip-value, as described in [14], we interpret $\overline{Z}_{a,b}$ as a probability distribution function and generate the corresponding empirical cumulative representation (ECDF). Next, we start to iterate over the ECDF to find the modal interval $[\overline{z}_l, \overline{z}_u], \overline{z}_l, \overline{z}_u \in \overline{Z}_{a,b}$, as well as the modal triangle $\Delta = ((\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N}), (\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N}), (\overline{z}_{\overline{m}_3}, \frac{\overline{m}_3}{N}))$, where $\overline{m}_1 \leq \overline{m}_2 \leq \overline{m}_3$ are the indices of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3} \in \overline{Z}_{a,b}$ respectively, i.e. the indices when considering the sorted projected data. The modal interval is the area within which the maximum slope of the ECDF lies. This corresponds to the most significant mode(s) of the underlying distribution. While the modal interval iteratively shrinks, the modal triangle introduced by [22] is the triangle formed by $(\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N})$, $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$ and $(\overline{z}_{\overline{m}_3}, \frac{\overline{m}_3}{N})$ that, given the modal interval, has the largest distance between the ECDF and a piecewise linear function that satisfies the conditions of unimodality (convex until $\overline{z}_l$ and concave after $\overline{z}_u$). A visualization of this process is given in Figure 2. The modal triangle fulfills $\text{height}(\Delta) = 2 \cdot \text{dip}$ (the details can be found in [22]). Therefore, the Dip-value can be calculated as follows:

$$\text{dip}(\overline{Z}_{a,b}) = \frac{1}{2N}\left(\overbrace{\left|\frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2\right|}^{=:A} + 1\right)$$

Using this formulation we can deduce the gradient regarding each feature i of the projection axis [22].

$$\frac{\partial \text{dip}(\overline{Z}_{a,b})}{\partial p_{a,b}[i]} = c\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \frac{(\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2})(z_{m_3}[i] - z_{m_1}[i])}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right), \tag{2}$$

where $c = \frac{\overline{m}_3 - \overline{m}_1}{2N}$ if $A > 0$, else $c = -(\frac{\overline{m}_3 - \overline{m}_1}{2N})$. Here, z_{m_j} is the (unprojected) m-dimensional embedded data point corresponding to $\overline{z}_{\overline{m}_j}$, i.e. $p_{a,b}^T \cdot z_{m_j} = \overline{z}_{\overline{m}_j}$, and $[i]$ indicates its i-th coordinate. Furthermore, we can calculate the gradient regarding each feature i of the samples.

$$\frac{\partial \text{dip}(\overline{Z}_{a,b})}{\partial z[i]} = \begin{cases} p_{a,b}[i]\, c \frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_1}, \\ p_{a,b}[i]\, c \frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} & \text{if } z = z_{m_2}, \\ p_{a,b}[i]\, c \frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_3}, \\ 0 & \text{else,} \end{cases} \tag{3}$$

where the definitions are the same as above. The derivations of these equations can be found in appendix D.

In order to clearly separate the clusters, we need to maximize the modality with respect to these samples. In mathematical terms this equals $\max(\text{dip}(\overline{Z}_{a,b}))$ or, since we use gradient descent, we minimize the negative value, i.e. $\min(-\text{dip}(\overline{Z}_{a,b}))$.

However, we must be careful that high modality is not achieved by tearing a cluster apart. Therefore, we need another term supporting a unimodal structure within the clusters on this projection axis. This is given by $\min(\text{dip}(\overline{Z}_{a,\not b}) + \text{dip}(\overline{Z}_{\not a,b}))$, where $\overline{Z}_{a,\not b} \subseteq \overline{Z}_{a,b}$ only contains the samples of cluster a. $\overline{Z}_{\not a,b}$ is defined analogously. We consider these two optimizations to be equally important, which is why we multiply the unimodal constraint by $\frac{1}{2}$.

Applying these terms to the batch-wise optimization of the DipEncoder yields the loss terms $\mathcal{L}_{uni}$ and $\mathcal{L}_{multi}$.

$$\mathcal{L}_{uni}(\mathcal{B}, a, b) = \frac{1}{2}\left(\text{dip}(\overline{Z}^{\mathcal{B}}_{a,\not b}) + \text{dip}(\overline{Z}^{\mathcal{B}}_{\not a,b})\right),$$

$$\mathcal{L}_{multi}(\mathcal{B}, a, b) = -\text{dip}(\overline{Z}^{\mathcal{B}}_{a,b}),$$

where $\overline{Z}^{\mathcal{B}}_{a,b} = \text{sort}\{p_{a,b}^T \cdot z | z \in \text{enc}(X_{a,b} \cap \mathcal{B})\}$. $\overline{Z}^{\mathcal{B}}_{a,\not b}$ and $\overline{Z}^{\mathcal{B}}_{\not a,b}$ are defined analogously. Combined we get the dip loss $\mathcal{L}_{dip}$.

$$\mathcal{L}_{dip}(\mathcal{B}) = \frac{2}{k(k-1)} \sum_{a=1}^{(k-1)} \sum_{b=a+1}^{k} \mathcal{L}_{uni}(\mathcal{B}, a, b) + \mathcal{L}_{multi}(\mathcal{B}, a, b)$$

With the previously defined gradients (Eq. 2 and 3), we can now optimize the DipEncoder by backpropagation based on these Dip-values. Here the gradient from the projection axes feeds into the DipModule, and the gradient with respect to the data is used to update the neurons of the AE.

One problem with the current loss function is that it does not generalize very well since the embedding is optimized based only on the modal triangle. That is, we only get a non-zero gradient for three samples per execution of the Dip-test. For this reason, we also include the reconstruction loss (Eq. 1) into our final loss term.

$$\mathcal{L}_{final}(\mathcal{B}) = \mathcal{L}_{dip}(\mathcal{B}) + \lambda \mathcal{L}_{rec}(\mathcal{B}) \tag{4}$$

We would like the reconstruction loss to be weighted similarly to $\mathcal{L}_{uni}$ and $\mathcal{L}_{multi}$. Due to the nature of the Dip-value these are limited to $(0, 0.25]$ and $[-0.25, 0)$ respectively. To constrain $\mathcal{L}_{rec}$, we need to define a hypothetical maximum value. For this we assume that the AE initially, i.e. before the backpropagation is executed for the first time, is in its worst state. We therefore set $\lambda = \frac{1}{4\mathcal{L}_{rec}(\mathcal{B}_{init})}$, where $\mathcal{B}_{init}$ is the first batch of data. This makes $0 \leq \lambda \mathcal{L}_{rec}(\mathcal{B}) \leq 0.25$ approximately valid for all $\mathcal{B}$.

To speed up the separation of the individual clusters, we multiply the gradients originating from the Dip-tests within $\mathcal{L}_{uni}$ by dip, where dip equals the corresponding Dip-value, and the gradient originating from the Dip-test within $\mathcal{L}_{multi}$ by $(0.25 - dip)$. This strategy reduces, for example, the weighting of $\mathcal{L}_{uni}$ concerning clusters that already indicate a unimodal structure. Thus, we shift the focus of the optimization towards structures that do not yet show the desired characteristics.

Note that the Dip-test only gives meaningful values if a certain amount of samples is present. Since our samples are divided into several clusters, we need larger batches with more clusters present. Based on an experimental analysis (appendix B), we recommend a minimum batch size of $25 \cdot k$, where k is the number of clusters.

As already shown in [25], the Dip-test struggles to notice a mode as such if it is significantly smaller than another one. Thus, a cluster containing many points together with a cluster containing significantly fewer points is still considered unimodal. This, in particular, weakens the significance of the modal triangle in $\mathcal{L}_{multi}$ and thus the usability of the gradient. To avoid this, [25] suggests to use only the $S|X_b|$ samples from cluster a that are closest to cluster b if $|X_a| > S|X_b|$. Since we want to preserve the general structure of

both clusters as much as possible, we utilize a variation of this idea by randomly sampling $3|\overline{Z}^{\mathcal{B}}_{\not\phi,b}|$ objects from $\overline{Z}^{\mathcal{B}}_{a,\not\phi}$ if $|\overline{Z}^{\mathcal{B}}_{a,\not\phi}| > 3|\overline{Z}^{\mathcal{B}}_{\not\phi,b}|$.

Figure 3 shows an execution of the DipEncoder (3(a) - 3(d)) on the Optdigits data set using the ground truth labels. We see that the clusters within the embedding of the DipEncoder adopt a unimodal structure that is clearly separated from other clusters. Compared to a regular AE (3(e) - 3(h)), the structures stand out much better.

One advantage of the DipEncoder is that we do not have to assume a distribution in advance. We only want to achieve a unimodal structure within a cluster and a multimodal structure between clusters. This is a perfect starting point for unsupervised learning algorithms since the inter- and intra-cluster dependencies are often unknown a priori.

3.1 Update the cluster labels

We use the components of the DipEncoder to develop a parameter-free method to update the cluster labels. For this, we utilize the property that the Dip-test does not only return the Dip-value and the modal triangle but also the modal interval $[\overline{z}_l, \overline{z}_u]$. This interval shows the area of the steepest slope in the ECDF and thus describes the most significant mode(s) within our samples. We use the intervals returned by $\mathcal{L}_{uni}$ to determine the areas of influence of the two clusters on their corresponding projection axis. This information can then be applied to define a threshold that indicates whether an object should be assigned to the left or the right cluster. The threshold T is simply the center point between the upper limit $\overline{z}_{u,L}$ of the left cluster L and the lower limit $\overline{z}_{l,R}$ of the right cluster R.

$$T = (\overline{z}_{u,L} + \overline{z}_{l,R})/2 \tag{5}$$

An example of this process can be seen in Figure 4. Initially, some objects to the right of T still belong to the yellow cluster. However, our procedure indicates that these should rather match the purple cluster. The same applies analogously to objects from the purple cluster that lie to the left of T.

If we want to update the cluster labels, we project each sample onto each projection axis $p_{a,b}$ and determine whether it belongs rather to the left or the right cluster. In the end, each object is assigned to the cluster with which it has most frequently matched. Usually, one cluster always matches, making the assignment unambiguous. If a tie does occur, the sample is assigned to the cluster with the lower ID.

The described method has one major shortcoming. Since the modal interval specifies the area of the steepest slope in the ECDF while satisfying the unimodality constraints (first convex, then concave), this range turns out to be very small in the case of an already unimodal distribution. This behavior is reasonable by definition, but it does not fully reflect our human understanding of the most significant mode. Therefore, we apply a strategy introduced in the implementation of [28]. By mirroring the data set, we can assume quite reliably that we obtain a multimodal structure. It follows that the resulting modal interval has a higher significance with respect to our application. Since we do not change the structure of the samples, we can transfer the mirrored interval to the non-mirrored case. A visual representation of this strategy is presented in Figure 5. We can see that the modal interval of the mirrored samples captures our natural perception of the mode much better.

Algorithm 1: Pseudocode of the DipEncoder

Input: data set X, number of clusters k, number of epochs E
Output: *labels*

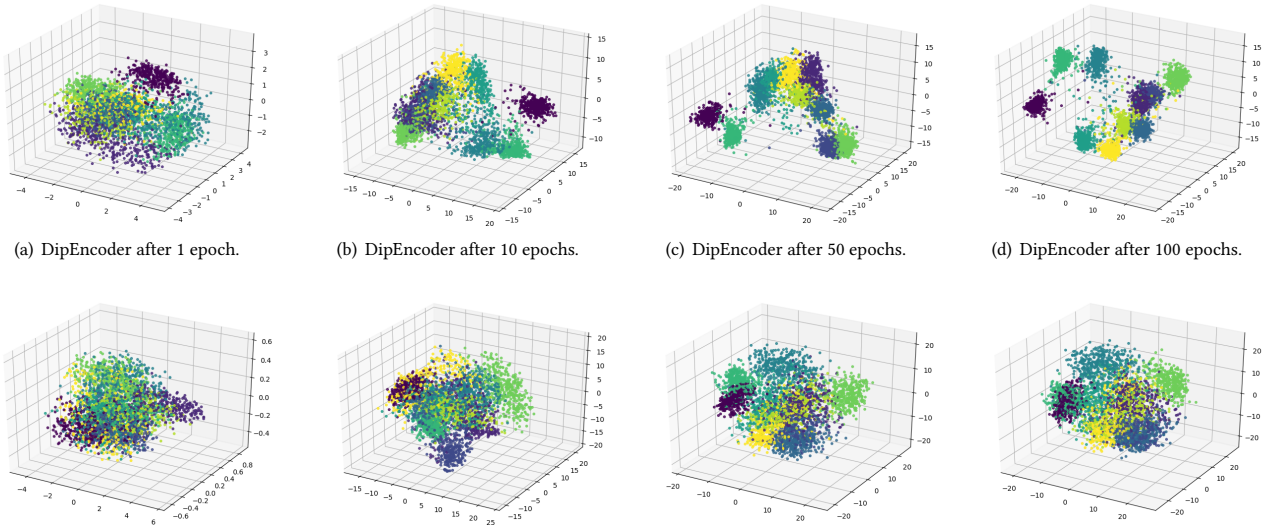
```
// Pretrain AE; save the reconstruction loss of B_init as λ
(AE, λ) = pretrain autoencoder on X using L_rec (Eq. 1)
// Get initial labels and projection axes
labels = k-means(AE.encode(X), k)
DM = DipModule(X, AE, labels) (Eq. 6)
for epoch = 0; epoch ≤ E; epoch += 1 do
    // Update labels as described in Section 3.1
    for x ∈ AE.encode(X) do
        clusterMatches = [0, ..., 0] ∈ ℝ^k
        for a = 1; a ≤ k − 1; a += 1 do
            for b = a + 1; b ≤ k; b += 1 do
                p_{a,b} = DM.getProjectionAxis(a, b)
                Z̄_{a,∤} = sort{p_{a,b}^T · z | z ∈ AE.encode(X_a)}
                Z̄_{∤,b} = sort{p_{a,b}^T · z | z ∈ AE.encode(X_b)}
                [z̄_{l,a}, z̄_{u,a}], [z̄_{l,b}, z̄_{u,b}] = dip(Z̄_{a,∤}), dip(Z̄_{∤,b})
                c_L, c_R = ids of left and right cluster on p_{a,b}
                T = center between the clusters (Eq. 5)
                if (p_{a,b}^T · x) < T then
                    clusterMatches[c_L] += 1
                else
                    clusterMatches[c_R] += 1
        set label of x to argmax(clusterMatches)
    if epoch == E then
        break
    // Train the DipEncoder
    for B in X do
        L_final = L_dip(B) + λ L_rec(B) (Eq. 4)
        optimize AE and DM using L_final
return labels
```

3.2 Clustering algorithm

Next, we utilize the concepts described above to develop a novel clustering algorithm. A pseudocode version of our approach is given in Algorithm 1.

Since the DipEncoder calculates the Dip-values based on known labels, initial clusters are required first. For this, we pretrain an AE using the reconstruction loss and then run an arbitrary clustering algorithm in the resulting embedding. For simplicity, we use k-means for this. We want to emphasize that other algorithms are easily applicable since we do not make any assumptions about distributions and only analyze modalities. The requirement of an initial clustering is often found in DC (for example in [7, 13, 17, 25, 38, 39]). Therefore, these methods can also be seen as AE-based cluster refinement methods.

Once we have initial cluster labels, we can create the DipModule. To obtain initial projection axes $p_{a,b}$, we calculate the distances between each combination of cluster centers within the embedding



(a) DipEncoder after 1 epoch. (b) DipEncoder after 10 epochs. (c) DipEncoder after 50 epochs. (d) DipEncoder after 100 epochs.

(e) AE after 1 epoch. (f) AE after 10 epochs. (g) AE after 50 epochs. (h) AE after 100 epochs.

Figure 3: The images show the embeddings ($m = 10$) of the DipEncoder (3(a) - 3(d)) and a regular AE (3(e) - 3(h)) for the Optdigits data set after 1, 10, 50 and 100 epochs. The colors indicate the clusters of the samples. To create a three-dimensional plot, we use the first three components of a PCA.

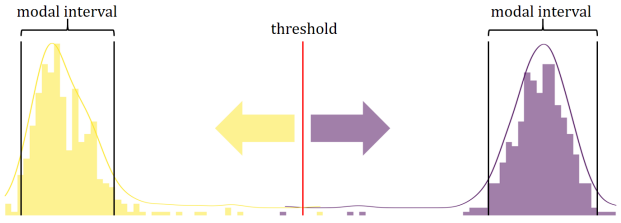


Figure 4: Illustration of our label update method using two clusters projected onto their corresponding projection axis. A threshold T is determined by calculating the center point between $\overline{z}_u$ of the yellow cluster (left) and $\overline{z}_l$ of the purple cluster (right). For each object, the relative position with respect to T is used to check which cluster fits better.

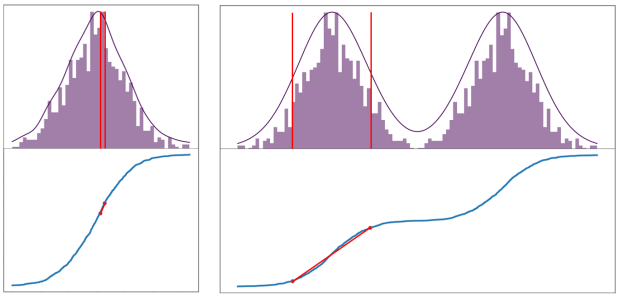
of the pretrained AE.

$$p_{a,b} = \frac{\sum_{x \in X_a} \text{enc}(x)}{|X_a|} - \frac{\sum_{x \in X_b} \text{enc}(x)}{|X_b|} \tag{6}$$

Now we start the iterative optimization of our clustering. First, for each cluster combination we compute the Dip-value with respect to $p_{a,b}$ and update the labels as described in Section 3.1. Next, we use the updated labels to calculate the loss function $\mathcal{L}_{final}$ (Eq. 4) in a batch-wise fashion and thus optimize the AE and DipModule. This procedure repeats for a predefined number of iterations.

4 EXPERIMENTS

To verify the quality of our approach, we conduct experiments on a variety of real-world data sets. Therefore, we compare our results to those of competitor algorithms.



(a) Non-mirrored samples. (b) Mirrored samples.

Figure 5: The images show the modal interval of an example distribution. [Top] Histogram of the data set. [Bottom] The corresponding ECDF. The modal interval is represented by a red line. 5(a) shows the original samples, while 5(b) shows the samples mirrored to the left.

Data sets: First, we would like to briefly describe the used data sets. The general information (N, d, k) can be found in the header of Table 1. To present the wide application field of our method, we use data sets from different domains. Optdigits [8], USPS [16] and MNIST [24] are image data sets containing the digits 0-9. The image data set Fashion-MNIST (F-MNIST) [37] shows items from the *zalando* online store and Kuzushiji-MNIST (K-MNIST) [3] consists of various Kanji characters. Human Activity Recognition (HAR) [8], Pendigits [8] and Letterrecognition (Letters) [8] are numerical data sets representing sensor records of human activity, coordinates of handwritten digits and letter stimuli, respectively. Furthermore, we

Table 1: NMI results of the DipEncoder against the comparison algorithms on various data sets. Each experiment is repeated ten times, and the average result ± standard deviation as well as the maximum result (in brackets) are stated in %. The best average and maximum result per data set are underlined, the runner-up is dotted.

Method	**Optdigits** ($k = 10$) ($N = 5620, d = 64$)	**USPS** ($k = 10$) ($N = 9298, d = 256$)	**HAR** ($k = 6$) ($N = 10299, d = 561$)	**Pendigits** ($k = 10$) ($N = 10992, d = 16$)	**Reuters10k** ($k = 4$) ($N = 10000, d = 2000$)
DipEncoder	88.6 ± 3.0 (92.0)	81.9 ± 0.8 (83.6)	73.5 ± 6.9 (82.4)	75.2 ± 2.1 (78.2)	36.8 ± 4.2 (41.9)
AE+k-means	80.1 ± 2.4 (83.8)	69.6 ± 0.9 (71.2)	67.5 ± 4.2 (73.2)	70.0 ± 0.8 (72.2)	37.2 ± 6.3 (47.3)
DEC	88.5 ± 2.5 (91.9)	80.7 ± 0.6 (81.4)	66.3 ± 4.8 (76.8)	76.9 ± 1.1 (77.9)	37.9 ± 7.1 (51.6)
IDEC	80.4 ± 2.4 (84.0)	69.3 ± 1.0 (70.9)	69.9 ± 2.9 (74.1)	69.8 ± 1.3 (72.2)	39.1 ± 6.7 (51.9)
DCN	84.8 ± 2.3 (84.8)	74.6 ± 1.3 (76.1)	73.4 ± 4.8 (80.9)	73.5 ± 0.5 (74.4)	35.1 ± 7.1 (45.4)
DipDECK	83.5 ± 2.3 (86.7)	68.5 ± 1.3 (70.5)	70.8 ± 1.3 (72.1)	72.8 ± 1.2 (74.7)	15.6 ± 18.1 (45.8)

Method	**20Newsgroups** ($k = 20$) ($N = 18846, d = 2000$)	**Letters** ($k = 26$) ($N = 20000, d = 16$)	**MNIST** ($k = 10$) ($N = 70000, d = 784$)	**F-MNIST** ($k = 10$) ($N = 70000, d = 784$)	**K-MNIST** ($k = 10$) ($N = 70000, d = 784$)
DipEncoder	30.8 ± 0.6 (31.6)	47.1 ± 0.9 (48.2)	85.8 ± 1.6 (87.8)	60.6 ± 2.2 (63.5)	52.2 ± 3.2 (57.0)
AE+k-means	31.2 ± 0.8 (32.4)	42.3 ± 0.9 (43.4)	74.4 ± 1.5 (77.0)	54.2 ± 0.5 (55.1)	46.3 ± 2.4 (49.7)
DEC	15.8 ± 1.0 (17.1)	46.0 ± 2.0 (48.0)	85.2 ± 1.2 (86.6)	59.7 ± 1.4 (62.5)	54.2 ± 2.1 (58.0)
IDEC	28.1 ± 1.3 (30.2)	45.1 ± 1.4 (47.6)	75.3 ± 1.2 (77.8)	54.3 ± 0.7 (55.3)	46.7 ± 1.8 (49.1)
DCN	28.6 ± 1.5 (30.7)	43.7 ± 2.4 (48.4)	82.0 ± 1.7 (84.2)	56.0 ± 0.9 (58.3)	48.2 ± 2.0 (51.7)
DipDECK	00.1 ± 0.0 (00.1)	34.5 ± 3.6 (38.2)	75.8 ± 2.0 (79.4)	53.9 ± 2.9 (57.2)	38.7 ± 4.1 (43.0)

consider the document data sets Reuters [26] and 20Newsgroups[1]. For 20Newsgroups we convert the documents to vectors with TF-IDF, using only the most common 2000 features. We preprocess Reuters as described in [38], using a subset of 10000 objects, called Reuters10k.

Apart from TF-IDF on the document data sets, we do not use other preprocessing steps. Since all DC algorithms are based on an AE, they should be able to learn relevant transformations on their own.

AE setup: We choose the same structure of the AE as presented in [38]. This corresponds to the AE dimensions $d - 500 - 500 - 2000 - m - 2000 - 500 - 500 - d$, where $m = 10$, which have already been used by other DC algorithms, like [13] or [17]. We would like to have a starting situation as similar as possible in our experiments. Therefore, ten AEs are pretrained for each data set, forming the initial setting for all DC algorithms. These AEs are trained for 100 epochs using the ADAM [20] optimizer with a constant learning rate of 0.001 and a batch size of 256. The actual clustering optimization then follows.

During clustering, the DipEncoder uses a learning rate of 0.0001 with a batch size of $25 \cdot k$ (as described in Section 3). The other DC algorithms use a learning rate of 0.0001 with a batch size of 256. All optimizations were run for another 100 epochs.

The DipEncoder was implemented using PyTorch[2] and can be downloaded at: https://dmm.dbs.ifi.lmu.de/downloads.

Comparison algorithms: As comparison algorithms, we choose those comparable by architecture since only these can load the pretrained AEs naturally. These algorithms are DEC, IDEC, DCN and DipDECK. In addition, we execute k-means on the pretrained AEs (AE+k-means). All parameters are set as described in the respective papers. The only exception is DipDECK, where we set $m = 10$ instead of 5. By comparing to the AE+k-means results, one can see how much the results have improved since all the procedures start from the same initial situation. An exception is DipDECk, since there the initial k-means execution is run with a highly overestimated number of clusters ($k_{init} = 35$).

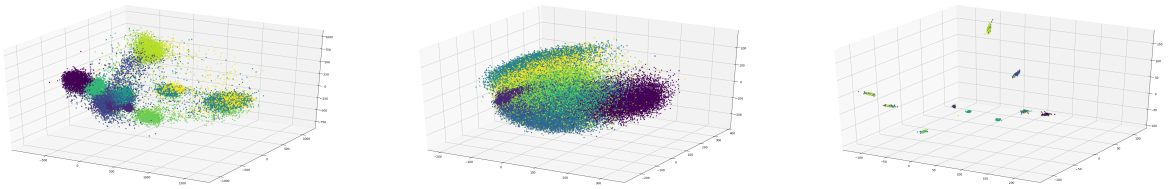
Metrics: We evaluate our approaches using the *normalized mutual information* (NMI) and the *adjusted rand index* (ARI). Both are established metrics in clustering that can take values between 0 and 1, where 1 indicates a perfect result. We report all results in %.

Evaluation: The NMI results of the experiments can be found in Table 1 (ARI results in appendix C). We can see that our method performs best in 12 out of 20 experiments (mean and max). In addition, we take second place six times. We generally show good results on both image and numerical data sets. Only concerning text data do the comparison algorithms perform slightly better. In this case, the other Dip-based clustering method, DipDECK, also seems to have significant problems. For example, it identifies only 1 or 2 clusters for 20Newsgroups, which explains the inferior results. From this, it can be concluded that text data sets, possibly due to the sparse structure, show less multimodal features.

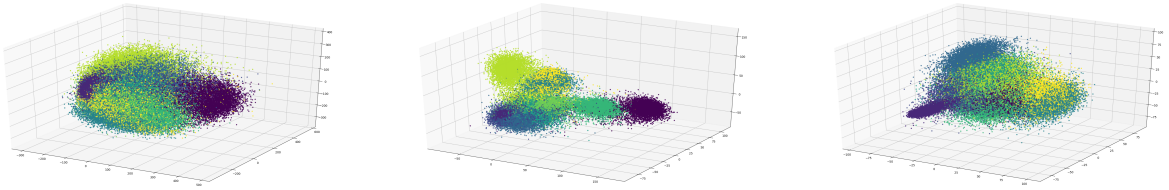
Figure 6 illustrates the results of the compared DC algorithms on MNIST. It is easy to see that the procedures operate with different degrees of rigor as far as the shifting of data is concerned. DEC offers the greatest flexibility since it does not apply any regularization term such as the reconstruction loss. Therefore, one can clearly notice the compressed clusters as identified by DEC. The DipEncoder seems to be the second most flexible algorithm in this regard. AE+k-means, on the other hand, is by definition not able to separate clusters from each other. IDEC and DipDECK limit the ability to push clusters apart by integrating the reconstruction loss into the cost function using a fixed weighting factor. This factor equals 10 compared to the cluster loss for IDEC and 1 for DipDECK.

We want to elaborate on these differences a bit more. When we compare our results from Table 1 with the results from other publications, some comparison methods seem to have problems

[1] http://qwone.com/~jason/20Newsgroups/
[2] https://pytorch.org/



(a) DipEncoder. (b) AE+k-means. (c) DEC.



(d) IDEC. (e) DCN. (f) DipDECK.

Figure 6: Images of the final embeddings ($m = 10$) of MNIST as created by different Deep Clustering algorithms. The colors depict the ground truth labels of the clusters. Features correspond to the first three components of a PCA.

Table 2: DC results on a standardized version of MNIST (zero mean and unit variance). Each experiment is again repeated ten times. Representation corresponds to Table 1.

Method	MNIST-Standardized ($k = 10$) ($N = 70000, d = 784$)	
	NMI	ARI
DipEncoder	85.3 ± 2.3 (88.2)	78.7 ± 4.8 (85.0)
AE+k-means	70.9 ± 2.7 (74.8)	63.4 ± 4.6 (70.1)
DEC	82.1 ± 2.6 (85.5)	75.7 ± 5.4 (82.4)
IDEC	85.2 ± 2.0 (87.4)	78.6 ± 4.4 (84.2)
DCN	81.8 ± 2.4 (84.8)	73.9 ± 5.0 (81.2)
DipDECK	81.6 ± 2.3 (86.0)	67.8 ± 6.2 (80.7)

if the data sets have not been standardized (zero mean and unit variance) as a preprocessing step. This concerns, in particular, those methods that use a constant factor for weighting the reconstruction loss without considering the scaling of the original space. To illustrate this, we conduct another experiment where we rerun all algorithms on a standardized version of MNIST. The experiments were performed as described above. The results are displayed in Table 2. We can see that while the results from the DipEncoder are almost unchanged, DipDECK and, in particular, IDEC perform significantly better. This shows the benefit of weighting the reconstruction loss depending on the scaling of the data set to achieve a value similar to the clustering loss.

An additional experiment is performed to verify that our method is able to generalize. Therefore, we apply the DipEncoder in a supervised manner. For this, we use the provided test-training partitions of MNIST and use the known training labels to optimize the DipEncoder. Here, we can skip the pretraining of the AE and the initial k-means execution. In this supervised version of the DipEncoder, the update of the labels is executed only for the test data, after the optimization of the training data is finished after 100 epochs. We compare our results with those of a support vector machine (SVM) [4] in combination with different dimension reduction techniques. SVM is a machine learning algorithm that tries to place planes in the feature space to separate the classes as well as possible. This procedure compares well with our approach as described in Section 3.1. We run SVM in combination with PCA, as the probably most used dimensionality reduction technique, LDA, as a frequently used supervised method, and an AE, as it is the basis of our approach. All methods reduce the number of features to 10 (using the implementations from scikit-learn[3]). Furthermore, we also perform SVM in the embedding created by the DipEncoder. This gives us conclusions about the quality of our label update method. To evaluate these supervised experiments, we use the test data not used for the prior optimization. We use the accuracy (ACC) as a common metric in supervised learning to quantify the results. It ranges from 0 (poor) to 1 (perfect). Additionally, the NMI values are given, so we are able to compare with the previous clustering results. The supervised results are shown in Table 3.

The analysis indicates that the DipEncoder generalizes very well, assigning even previously unknown samples into the correct groups in most cases. Since the supervised DipEncoder also produces better results than SVM in combination with the embedding of the DipEncoder, we can assume that our procedure to update the cluster labels defines reasonable bounds for the clusters.

5 CONCLUSION

In this paper, we have shown for the first time how to calculate the gradient of the Dip-value with respect to the data. To demonstrate how this can practically be used, we developed the DipEncoder. It is an extension of an autoencoder that uses the Dip-test to separate samples from different clusters in an embedding. The only condition is that samples of one cluster show a unimodal structure while the

[3] https://scikit-learn.org/stable/index.html

Table 3: Results of a supervised version of the DipEncoder against SVM in combination with different dimensionality reduction techniques. Each experiment is again repeated ten times. Representation corresponds to Table 1.

Method	**MNIST** ($k = 10$) ($^{N_{\text{train}} = 60000}_{N_{\text{test}} = 10000}$, $d = 784$)	
	ACC	NMI
DipEncoder$_{supervised}$	94.2 ± 3.9 (97.2)	90.5 ± 2.3 (92.7)
DipEncoder+SVM	92.9 ± 4.3 (97.1)	88.1 ± 3.1 (92.5)
SVM	86.6 ± 1.1 (87.5)	74.5 ± 1.1 (75.5)
PCA+SVM	58.5 ± 4.9 (67.7)	45.9 ± 3.4 (53.9)
LDA+SVM	87.7 ± 0.0 (87.7)	74.7 ± 0.0 (74.7)
AE+SVM	89.1 ± 1.7 (91.2)	79.4 ± 2.0 (81.8)

combined samples of two clusters show a multimodal structure. A previously defined distribution or other cluster properties are not necessary.

Further, we developed a Dip-based method that can update the labels within each iteration of the DipEncoder. The resulting deep clustering algorithm shows state-of-the-art results on various real-world data sets.

We think that the gradient of the Dip-value with respect to the data allows for a series of new research opportunities. In particular, we expect further research to integrate established Dip-based clustering methods to determine the number of clusters automatically.

ACKNOWLEDGMENTS

This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

The present contribution is supported by the Helmholtz Association under the joint research school 'Munich School for Data Science - MUDS'.

REFERENCES

[1] Dana H Ballard. 1987. Modular learning in neural networks.. In *AAAI*, Vol. 647. 279–284.
[2] Theofilos Chamalis and Aristidis Likas. 2018. The Projected Dip-means Clustering Algorithm. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN*. ACM, 14:1–14:7.
[3] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. 2018. Deep Learning for Classical Japanese Literature. *CoRR* (2018).
[4] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (1995), 273–297.
[5] Ying Cui, Xiaoli Z. Fern, and Jennifer G. Dy. 2007. Non-redundant Multi-view Clustering via Orthogonalization. In *ICDM*. IEEE Computer Society, 133–142.
[6] Chris H. Q. Ding and Tao Li. 2007. Adaptive dimension reduction using discriminant analysis and *K*-means clustering. In *ICML (ACM International Conference Proceeding Series, Vol. 227)*. ACM, 521–528.
[7] Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. 2017. Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *ICCV*. IEEE Computer Society, 5747–5756.
[8] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[9] R. A. FISHER. 1938. THE STATISTICAL UTILIZATION OF MULTIPLE MEASUREMENTS. *Annals of Eugenics* 8, 4 (1938), 376–386.
[10] Karl Pearson F.R.S. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.
[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. 2672–2680.
[12] Wengang Guo, Kaiyan Lin, and Wei Ye. 2021. Deep Embedded K-Means Clustering. In *ICDM*. IEEE, 686–694.
[13] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved Deep Embedded Clustering with Local Structure Preservation. In *IJCAI*. ijcai.org, 1753–1759.
[14] John A Hartigan and Pamela M Hartigan. 1985. The dip test of unimodality. *The annals of Statistics* (1985), 70–84.
[15] PM Hartigan. 1985. Computation of the dip statistic to test for unimodality: Algorithm as 217. *Applied Statistics* 34, 3 (1985), 320–5.
[16] Jonathan J. Hull. 1994. A Database for Handwritten Text Recognition Research. *IEEE Trans. Pattern Anal. Mach. Intell.* 16, 5 (1994), 550–554.
[17] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *IJCAI*. ijcai.org, 1965–1972.
[18] Christian Jutten and Jeanny Hérault. 1991. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Process.* 24, 1 (1991), 1–10.
[19] Argyris Kalogeratos and Aristidis Likas. 2012. Dip-means: an incremental clustering method for estimating the number of clusters. In *NIPS*. 2402–2410.
[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[21] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.
[22] Andreas Krause and Volkmar Liebscher. 2005. Multimodal projection pursuit using the dip statistic. (2005).
[23] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[25] Collin Leiber, Lena G. M. Bauer, Benjamin Schelling, Christian Böhm, and Claudia Plant. 2021. Dip-based Deep Embedded Clustering with k-Estimation. In *ACM SIGKDD*. ACM, 903–913.
[26] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.* 5 (2004), 361–397.
[27] Yonghong Luo, Ying Zhang, Xiaoke Ding, Xiangrui Cai, Chunyao Song, and Xiaojie Yuan. 2018. StrDip: A Fast Data Stream Clustering Algorithm Using the Dip Test of Unimodality. In *WISE (Lecture Notes in Computer Science, Vol. 11234)*. Springer, 193–208.
[28] Samuel Maurus and Claudia Plant. 2016. Skinny-dip: Clustering in a Sea of Noise. In *ACM SIGKDD*. ACM, 1055–1064.
[29] Dominik Mautz, Claudia Plant, and Christian Böhm. 2019. Deep Embedded Cluster Tree. In *ICDM*. IEEE, 1258–1263.
[30] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. 2020. Non-Redundant Subspace Clusterings with Nr-Kmeans and Nr-DipMeans. *ACM Trans. Knowl. Discov. Data* 14, 5 (2020), 55:1–55:24.
[31] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.
[32] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. 2019. ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. In *AAAI*. AAAI Press, 4610–4617.
[33] Benjamin Schelling, Lena Greta Marie Bauer, Sahar Behzadi, and Claudia Plant. 2020. Utilizing Structure-Rich Features to Improve Clustering. In *ECML PKDD (Lecture Notes in Computer Science, Vol. 12457)*. Springer, 91–107.
[34] Benjamin Schelling and Claudia Plant. 2018. DipTransformation: Enhancing the Structure of a Dataset and Thereby Improving Clustering. In *ICDM*. IEEE Computer Society, 407–416.
[35] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.
[36] Herman Wold. 1966. Estimation of principal components and related models by iterative least squares. *Multivariate analysis* (1966), 391–420.
[37] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* (2017).
[38] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In *ICML (JMLR Workshop and Conference Proceedings, Vol. 48)*. JMLR.org, 478–487.
[39] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. 2017. Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 3861–3870.
[40] Jianwei Yang, Devi Parikh, and Dhruv Batra. 2016. Joint Unsupervised Learning of Deep Representations and Image Clusters. In *CVPR*. IEEE Computer Society, 5147–5156.

APPENDIX

In the appendix, we give an overview of the used symbols, add ARI results for our quantitative experiments, justify our chosen batch size, and present the derivations of the Dip-value and its gradients.

A SYMBOLS

All symbols used in the paper are explained in Table 4.

Table 4: Overview of the used symbols.

Symbol	Description
$N \in \mathbb{N}$	Size of the data set
$d \in \mathbb{N}$	Dimensionality of the original feature space
$m \in \mathbb{N}$	Dimensionality of the embedded space
$k \in \mathbb{N}$	Number of clusters
$T \in \mathbb{R}$	Threshold of the label update method
$X \subseteq \mathbb{R}^d$	Set of all objects
$\mathcal{B} \subseteq X$	One batch of data
$X_a \subseteq X$	Objects assigned to cluster a
$Z \subseteq \mathbb{R}^m$	Embedded set of all objects
	$\Rightarrow Z = \text{enc}(X)$
$Z_{a,b} \subseteq \mathbb{R}^m$	Embedded objects assigned to cluster a or b
	$\Rightarrow Z_{a,b} = \text{enc}(X_a \cup X_b)$
$p_{a,b} \in \mathbb{R}^m$	Projection axis of cluster a and b
$\overline{Z}_{a,b} \subseteq \mathbb{R}^1$	Sorted version of $Z_{a,b}$ projected to $p_{a,b}$
	$\Rightarrow \overline{Z}_{a,b} = \text{sort}\{p_{a,b}^T \cdot z \mid z \in Z_{a,b}\}$
$\overline{Z}_{a,\not b} \subseteq \overline{Z}_{a,b}$	Subset of $\overline{Z}_{a,b}$, ignoring samples from cluster b
$dip \in (0, 0.25]$	Dip-value (result of the Dip-test)
Δ	The modal triangle
$\overline{z}_l, \overline{z}_u \in \overline{Z}_{a,b}$	Lower and upper bound of the modal interval
$\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3} \in \overline{Z}_{a,b}$	x-coordinates of the modal triangle in the ECDF
$\overline{m}_1, \overline{m}_2, \overline{m}_3 \in \mathbb{N}$	Indices of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3}$
$z_{m_1}, z_{m_2}, z_{m_3} \in Z_{a,b}$	Non-projected representations of $\overline{z}_{\overline{m}_1}, \overline{z}_{\overline{m}_2}, \overline{z}_{\overline{m}_3}$
	$\Rightarrow \overline{z}_{\overline{m}_j} = p_{a,b}^T \cdot z_{m_j}$

B INFLUENCE OF THE BATCH SIZE

We would like to demonstrate by an experiment why an increasing batch size is necessary for the DipEncoder when the number of clusters increases. For this purpose, we execute the DipEncoder with batch sizes between $5 \cdot k$ and $50 \cdot k$ on different data sets. One should keep in mind that the number of clusters in the data sets ranges from 4 (Reuters10k) to 26 (Letters). The results can be seen in Figure 7.

Here we can observe that most records reach a plateau at around $15 \cdot k$. Above $25 \cdot k$, almost no improvements take place, which is why we select this factor as the default value.

C ARI RESULTS

In addition to the NMI results as shown in Table 1, we evaluate the experiments using the *adjusted rand index* (ARI). Those values are shown in Table 5.

D DERIVATIONS

To consolidate the understanding of our methods, we would like to present some derivations. First, we will deal with the calculation of the Dip-value before we take a closer look at the gradients with respect to the projection axis and the modal triangle. All symbols are defined as described above.

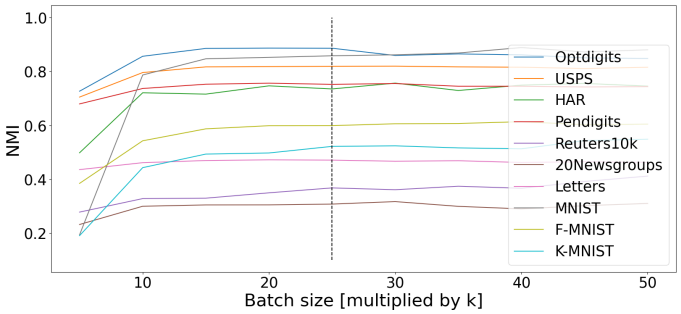


Figure 7: Results of the DipEncoder using different batch sizes on multiple data sets. Each entry corresponds to the average of 10 executions. The black vertical line illustrates the selected default value for the DipEncoder.

D.1 Derivation of the Dip-value

It has already been described in Section 3 that the Dip-value is determined based on the height of the modal triangle Δ on the ECDF. Following holds:

$$\text{height}(\Delta) = 2 \cdot \text{dip}(\overline{Z})$$

To solve this equation we determine the intersection between the vertical line towards $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$ and the line between $(\overline{z}_{\overline{m}_1}, \frac{\overline{m}_1}{N})$ and $(\overline{z}_{\overline{m}_3}, \frac{\overline{m}_3}{N})$. Note that the intersection can be above or below $(\overline{z}_{\overline{m}_2}, \frac{\overline{m}_2}{N})$.

$$\begin{pmatrix} \overline{z}_{\overline{m}_2} \\ \frac{\overline{m}_2}{N} \end{pmatrix} \pm \alpha \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \overline{z}_{\overline{m}_1} \\ \frac{\overline{m}_1}{N} \end{pmatrix} + \beta \begin{pmatrix} \overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1} \\ \frac{\overline{m}_3 - \overline{m}_1}{N} \end{pmatrix},$$

where $\alpha = \text{height}(\Delta)$.

$$\Rightarrow \beta = \frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}$$

$$\Rightarrow \alpha = \pm \frac{1}{N}(\overline{m}_1 - \overline{m}_2 + \frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}})$$

This yields:

$$\text{dip'}(\overline{Z}) = \frac{1}{2N}\left| \frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2 \right|$$

Krause et al. [22] add another $\frac{1}{2N}$ to this formula. This value was also used in the implementation outline from [15]. We assume a regularization purpose in the case of a degenerate triangle resulting in a Dip-value of 0, which is out of the bounds for the Dip-value. Note that this additional value has only a small influence on the final Dip-value and, as a constant, no impact on the gradients.

$$\text{dip}(\overline{Z}) = \frac{1}{2N}\left(\overbrace{\left| \frac{(\overline{m}_3 - \overline{m}_1)(\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_1})}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \overline{m}_1 - \overline{m}_2 \right|}^{=:A} + 1 \right)$$

Table 5: ARI results of the DipEncoder against the comparison algorithms on various data sets. Each experiment is repeated ten times. Representation corresponds to Table 1 (showing the NMI results).

Method	**Optdigits** ($k = 10$) ($N = 5620, d = 64$)	**USPS** ($k = 10$) ($N = 9298, d = 256$)	**HAR** ($k = 6$) ($N = 10299, d = 561$)	**Pendigits** ($k = 10$) ($N = 10992, d = 16$)	**Reuters10k** ($k = 4$) ($N = 10000, d = 2000$)
DipEncoder	85.6 ± 5.8 (91.4)	74.2 ± 1.1 (76.2)	63.4 ± 7.4 (73.5)	64.7 ± 4.2 (70.2)	35.4 ± 4.5 (43.4)
AE+k-means	76.7 ± 4.5 (83.0)	59.7 ± 1.4 (61.6)	58.7 ± 5.4 (65.1)	60.5 ± 1.9 (63.9)	39.9 ± 10.7 (56.7)
DEC	84.9 ± 5.2 (91.1)	72.7 ± 0.9 (74.1)	53.4 ± 7.7 (71.8)	66.7 ± 2.6 (68.6)	35.3 ± 10.6 (57.6)
IDEC	77.1 ± 4.6 (83.1)	59.3 ± 1.3 (60.9)	58.1 ± 2.8 (62.5)	60.2 ± 2.9 (63.9)	36.4 ± 10.1 (56.4)
DCN	81.1 ± 5.0 (86.2)	64.4 ± 2.3 (67.0)	62.7 ± 5.8 (71.9)	62.6 ± 2.0 (64.7)	29.6 ± 10.1 (49.8)
DipDECK	79.6 ± 5.5 (85.7)	58.7 ± 2.8 (63.1)	49.9 ± 1.1 (50.9)	61.7 ± 2.6 (65.4)	12.1 ± 14.3 (36.9)
Method	**20Newsgroups** ($k = 20$) ($N = 18846, d = 2000$)	**Letters** ($k = 26$) ($N = 20000, d = 16$)	**MNIST** ($k = 10$) ($N = 70000, d = 784$)	**F-MNIST** ($k = 10$) ($N = 70000, d = 784$)	**K-MNIST** ($k = 10$) ($N = 70000, d = 784$)
DipEncoder	18.0 ± 0.9 (19.2)	22.8 ± 0.9 (24.0)	81.0 ± 3.0 (84.5)	44.8 ± 2.8 (47.4)	37.7 ± 3.7 (43.9)
AE+k-means	16.9 ± 0.7 (17.9)	18.8 ± 0.6 (19.9)	69.1 ± 2.3 (73.2)	38.3 ± 1.1 (39.9)	32.4 ± 3.1 (37.9)
DEC	5.4 ± 0.6 (6.1)	20.5 ± 2.3 (23.4)	81.6 ± 2.1 (83.5)	44.0 ± 2.8 (48.9)	39.0 ± 2.3 (42.3)
IDEC	12.8 ± 1.1 (14.8)	21.3 ± 1.5 (23.6)	70.3 ± 2.0 (74.2)	38.1 ± 1.1 (39.9)	32.5 ± 2.2 (36.5)
DCN	15.1 ± 1.1 (17.4)	18.9 ± 2.5 (23.9)	77.1 ± 3.5 (80.8)	38.5 ± 1.2 (41.3)	31.5 ± 2.9 (37.5)
DipDECK	00.0 ± 0.0 (00.0)	7.2 ± 1.8 (9.1)	70.7 ± 2.5 (74.4)	32.8 ± 3.3 (37.6)	22.1 ± 4.0 (29.0)

D.2 Derivation of the gradients

Let us assume that $A > 0$ and consolidate factors that are irrelevant for the calculation of the gradients (if $A < 0$, all gradients must be multiplied by -1.). Following holds:

$$\begin{aligned}\mathrm{dip}(\overline{Z}) &= \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_2}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} - \frac{\overline{z}_{\overline{m}_1}}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}\right) + \text{const} \\ &= \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2}}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_1}}{p^T z_{m_3} - p^T z_{m_1}}\right) + \text{const}\end{aligned}$$

D.2.1 Gradient regarding the projection axis.

$$\begin{aligned}\frac{\partial}{\partial p[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N} \\ &\left(\left(\frac{z_{m_2}[i]}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_2}(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) - \right. \\ &\left.\left(\frac{z_{m_1}[i]}{p^T z_{m_3} - p^T z_{m_1}} - \frac{p^T z_{m_1}(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right)\right) \\ =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{p^T z_{m_3} - p^T z_{m_1}} + \right. \\ &\left.\frac{(p^T z_{m_1} - p^T z_{m_2})(z_{m_3}[i] - z_{m_1}[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{z_{m_2}[i] - z_{m_1}[i]}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} + \frac{(\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2})(z_{m_3}[i] - z_{m_1}[i])}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

D.2.2 Gradient regarding the modal triangle.

Considering z_{m_1}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_1}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{(-p^T z_{m_2})(-p[i])}{(p^T z_{m_3} - p^T z_{m_1})^2} - \right. \\ &\left.\left(\frac{p[i]}{p^T z_{m_3} - p^T z_{m_1}} + \frac{(-p^T z_{m_1})(-p[i])}{(p^T z_{m_3} - p^T z_{m_1})^2}\right)\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2} - p^T z_{m_1}}{(p^T z_{m_3} - p^T z_{m_1})^2} - \frac{1}{p^T z_{m_3} - p^T z_{m_1}}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_2} - p^T z_{m_1} - p^T z_{m_3} + p^T z_{m_1}}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

Considering z_{m_2}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_2}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p[i]}{p^T z_{m_3} - p^T z_{m_1}}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}}\right)\end{aligned}$$

Considering z_{m_3}:

$$\begin{aligned}\frac{\partial}{\partial z_{m_3}[i]}\mathrm{dip}(\overline{Z}) =& \frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{(-p^T z_{m_2})p[i]}{(p^T z_{m_3} - p^T z_{m_1})^2} - \frac{(-p^T z_{m_1})p[i]}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{p^T z_{m_1} - p^T z_{m_2}}{(p^T z_{m_3} - p^T z_{m_1})^2}\right) \\ =& p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\left(\frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2}\right)\end{aligned}$$

In summary:

$$\frac{\partial}{\partial z[i]}\mathrm{dip}(\overline{Z}) = \begin{cases} p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{\overline{z}_{\overline{m}_2} - \overline{z}_{\overline{m}_3}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_1}, \\ p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{1}{\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1}} & \text{if } z = z_{m_2}, \\ p[i]\frac{(\overline{m}_3 - \overline{m}_1)}{2N}\frac{\overline{z}_{\overline{m}_1} - \overline{z}_{\overline{m}_2}}{(\overline{z}_{\overline{m}_3} - \overline{z}_{\overline{m}_1})^2} & \text{if } z = z_{m_3}, \\ 0 & \text{else} \end{cases}$$