Chapter 6

Deep Learning of Point Clouds

Contents In the preceding chapters we have used sampling (MCMC methods) to perform inference. We extend here our point cloud datasets in 2D to much larger point cloud datasets in 3D. This requires a speed up in our inference methods.

Outline We introduce deep learning, and in particular variational autoencoders, ordinary autoencoders, sparse autoencoders, and convolutional autoencoders (Section 6.2). We show how they perform well on the MNIST dataset. We also show they do not perform well on the task of reconstructing 2D lines. We then introduce an autoencoder based on a model known in the literature as PointNet (Section 6.3). This autoencoder uses earth mover's distance (EMD) to reconstruct dense point clouds of single objects. In contrast, our dataset contains multiple objects that are sparse, such as squares and cubes. We show that the autoencoder does not learn a proper latent representation for those objects. We introduce two new metrics, the shifted earth mover's distance (SEMD) and the partitioning earth mover's distance (PEMD), to be used for datasets with objects on unknown positions or datasets with multiple objects in a single sample (Section 6.4). We test the new EMD on our dataset with multiple 3D cubes on different locations (Section 6.5). Finally, we provide the chapter conclusions and we describe some ways to improve structured autoencoders (Section 6.6).

6.1 Data-driven Methods

The previous chapter concluded with a suggestion to look into data-driven methods to accelerate inference even further than with the models described until then. One particular way in which we incorporate knowledge about the data we operate on, is by introducing nonlinear functions before we perform inference. These nonlinear functions we adapt to the data by a learning process. A neural network can represent such a function. If a stack of neural networks is used, this becomes the field of deep learning (LeCun et al., 2015; Schmidhuber, 2015).

An autoencoder consists of two of those deep neural networks that each contains multiple layers. The encoder has layers of decreasing size and maps to the code, a layer of latent variables. The decoder has layers of increasing size and maps from the code to the same dimension as the input. A loss function can be used to quantity the difference between the input of the autoencoder and its output. The calculated loss is used to adjust the weights in the neural networks through error propagation.

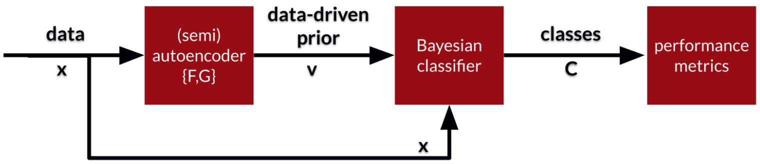


Figure 6.1: Left: the data x is used to train a (semi)autoencoder. Middle: the results of the autoencoder are used to create a data-driven prior for a Bayesian classifier. Right: the performance is measured with performance metrics like purity, rand index, as in previous chapters.

In Fig. 6.1 the autoencoder is embedded in a larger architecture The first block depicts an autoencoder that can efficiently represent 3D point cloud data. The second gets information from the autoencoder and uses this as a data-driven prior to perform classification. Typically, it is a Bayesian classifier as encountered in the previous chapters. The third block defines the performance of the autoencoder-classifier tandem.

In Section 6.2 we describe different types of autoencoders. In Section 6.3 we describe the earth mover's distance (EMD), a loss that can be used for the robotic vision domain of point clouds. We show the results of using this loss on the robotic vision task of fitting 3D cubes. We also visualize the latent representation of the autoencoder using this loss. In Section 6.4 we introduce a semi-autoencoder and introduce two generalizations of the EMD, the shifted earth mover's distance (SEMD) and the partitioning earth mover's distance (PEMD). In Section 6.5 we use the PEMD in combination with the triadic sampler of the previous chapter to perform inference over point clouds consisting of 3D cubes. We provide the chapter conclusions in Section 6.6.

6.2 Autoencoders

We introduce four autoencoders: a variational autoencoder (Section 6.2.1), an ordinary autoencoder (Section 6.2.2), a sparse autoencoder (Section 6.2.3), and a convolutional autoencoder (Section 6.2.4). We show how they perform on the MNIST dataset. We also show how they perform on a second dataset used in the previous chapter with 2D lines made out of 2D points. We will refer to the latter dataset as Lines100.

6.2.1 Variational Autoencoder

Variational autoencoders (Kingma and Welling, 2014; Rezende et al., 2014) are ordinary autoencoders with additional constraints on the latent variables. The latent variables in autoencoder parlance are called the code. In a variational autoencoder the latent variables are forced to approximately describe a standard Normal (or unit Gaussian) distribution. The autoencoder is trained using a loss function that is composed out of (1) a generative loss, a mean squared error that measures how accurately the network reconstructs its input, and (2) a latent loss, a KL-divergence that measures how closely the latent variables match a unit Gaussian. This loss is summed over all samples (and reconstructions), $L = \sum_i l_i(F,G)$.

$$l_i(F,G) = -\mathbb{E}_{h \sim q_F(h|x_i)}[\log(p_G(x_i|h)] + \mathbb{KL}(q_F(h|x_i)||p(h)) \tag{6.1}$$

To optimize the KL divergence a reparameterization trick is applied. The encoder does not generate a vector with real values, but generates a vector with means and standard deviations instead.

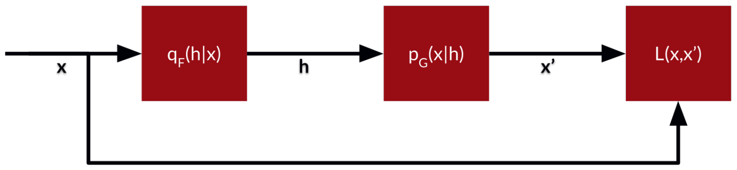


Figure 6.2: Left: $q_F(h|x)$ maps the data x to (hidden) random variables h. Middle: $p_G(x|h)$ maps the hidden random variables to reconstructed data x'. Right: $L(x,x')$ measures the similarity between x and x'.

The results are presented in the following manner. First, we visually inspect the reconstruction of the items in the dataset. Second, the test samples are encoded into the latent variable representation. The latent variables are then presented in a 2D scatterplot. Third, there is a sweep over the latent variable values to generate digits. The second and third presentations are especially useful if the encoder has only two latent variables. In that case the presentation in a 2D scatterplot does not require a dimensionality reduction step. The sweep over only two latent variables is also very easy to represent in 2D.

The MNIST digits are reconstructed by a variational autoencoder as shown in Fig. 6.3.

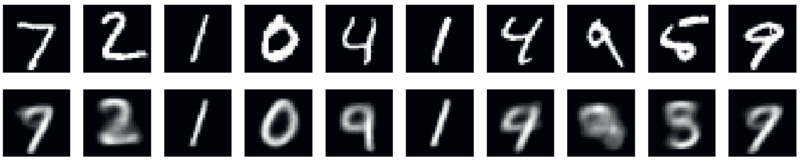


Figure 6.3: Reconstruction of MNIST data by a variational autoencoder. Top: the input, images with single hand-drawn digits. Bottom: the output, the images reconstructed by the autoencoder. During the training process the network weights are adjusted precisely so that the reconstruction loss between the output and the input is minimized. The variational autoencoder minimizes at the same time also the KL-divergence between the latent variables and a prior.

The scatterplot of the latent variable representation of the test set. It can be seen that similar digits are mapped to similar values in the latent space.

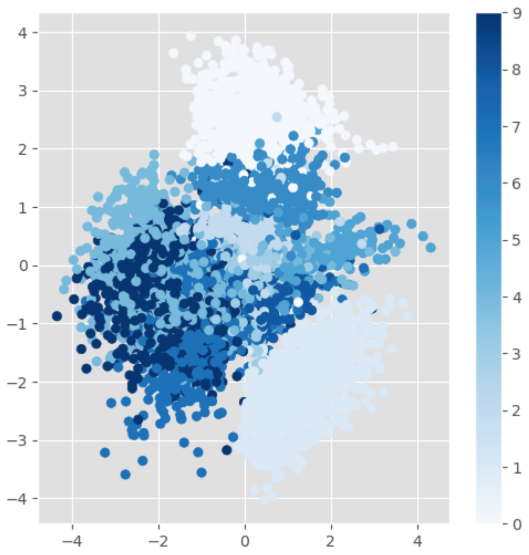


Figure 6.4: Scatterplot of latent variable representations of test samples in a variational autoencoder. There are two latent variables. The values of one latent variable are on the horizontal axis. The values of the other latent variables are on the vertical axis. The digits of the MNIST task are plotted with different color shades. For example, the digit zero is represented by values of around 0 of the "horizontal" latent variable and values around 3 of the "vertical" latent variable. Those are the whitest dots. The variables do not necessarily have (easily identifiable) semantics and are therefore not labeled.

Note that not every digit occupies the same amount of space in the latent variable layer. The amount of space emerges from the learning process.

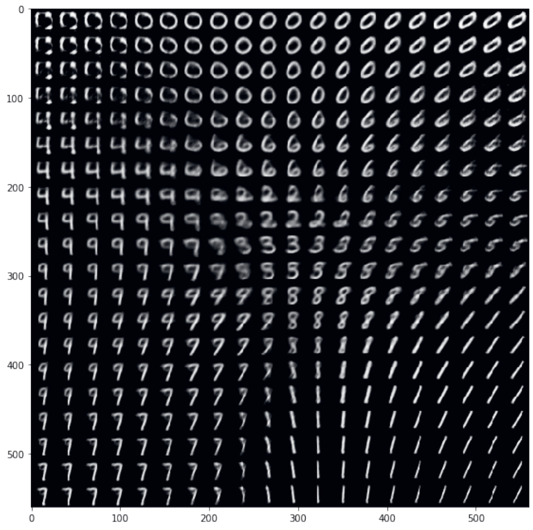


Figure 6.5: Latent variable sweep of test samples in a variational autoencoder. The two latent variables are both given values and the decoder calculates the output. This output is depicted as an image (a reconstructed "digit"). Here you see 20x20 images reconstructed starting from values at the top-left of around $(0,0)$ to the bottom-right of around $(560,560)$. It is clearly visible that if the values of the latent variables change only slightly, that the reconstruction is also changed only slightly.

6.2.2 Ordinary Autoencoder

An "ordinary" autoencoder (Rumelhart et al., 1986) has been trained with a latent variable layer of 32 nodes (rather than 2 as in the variational autoencoder above). It only minimize the reconstruction error and has no constraints[1] on the latent layers. We show the representation of the ordinary autoencoder after that of the variational autoencoder, so we can see how the representation is less structured.

$$l_i(F,G) = -\mathbb{E}_{h \sim q_F(h|x_i)}[\log p_G(x_i|h)] \tag{6.2}$$

The reconstruction is similar to that of the variational autoencoder if we just use visual inspection (see Fig. 6.6). This means that the ordinary autoencoder learns to reconstruct the digits just as the variational autoencoder.

[1]The size of the layer can be seen as a constraint, but it is not yet a regularization technique.

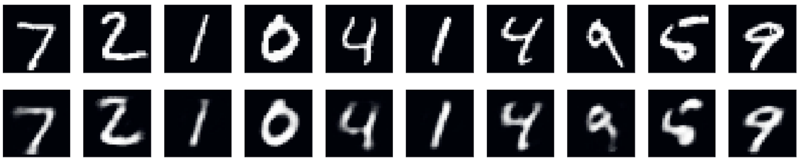


Figure 6.6: Reconstruction of MNIST data by an ordinary autoencoder.

The difference between the ordinary and the variational autoencoder shows when we start to study the latent representation. If we use a scatterplot for two of the latent variable nodes, there is not much structure to observe (see Fig. 6.7).

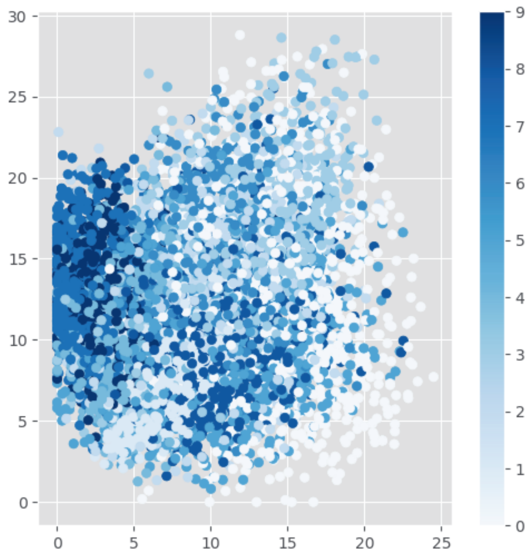


Figure 6.7: Scatterplot of latent variable representations of test samples in a ordinary autoencoder. The two axes represent only two nodes of in total 32 nodes in the latent layer. In contrast to the variational autoencoder (Fig. 6.4) there are no easily distinguishable clusters representing particular digits.

We might perform dimensionality reduction and for example use t-SNE (Van Der Maaten et al., 2009) to map to a 2D space. However, this is much more indirect than in the case that there are only two latent variables. If there is still no structure observed, it might be just an artifact of how t-SNE performs dimensionality reduction (not indicating the quality of the latent variable representation).

Let us use the ordinary autoencoder to reconstruct point clouds. In this case the reconstruction of 2D lines.

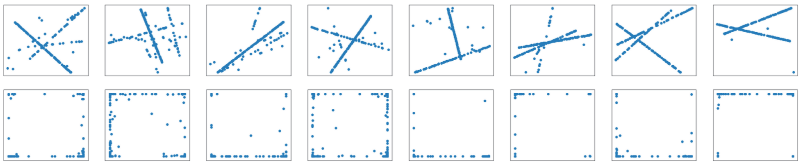


Figure 6.8: Top: the input, multiple lines per sample. Bottom: the output, the reconstructed 2D point cloud by the autoencoder. Clearly, the reconstruction fails for the ordinary autoencoder. In particular, the autoencoder seems not to be able to reconstruct the correlations between the x and y coordinates.

In Fig. 6.8 the reconstruction of the 2D lines are shown. The ordinary autoencoder is not able to reconstruct the lines.

6.2.3 Sparse Autoencoder

A sparse autoencoder (Hosseini-Asl et al., 2015) is similar to the ordinary autoencoder, but enforces sparsity through an "activity regularizer":

$$l_i(F,G) = -\mathbb{E}_{h\sim q_F(h|x_i)}[\log p_G(x_i|h)] + \mathbb{KL}\left(q_F(h|x_i)||p(h)\right) + J(h). \tag{6.3}$$

The activation in the hidden layer can be regularized by an L1 loss function, $J(h) = \lambda \sum_j |h_j|$, or by enforcing the average activation of a node to be close to zero (averaged over all m training samples). This can be achieved for example by setting $J(h) = \beta \sum_j \mathbb{KL}(\rho||\hat{\rho}_j)$ with $\hat{\rho}_j = \frac{1}{m}\sum_r^m [h_j(x_r)]$. The sharpness of the reconstructions can be tuned by the factors λ or β or both if both regularizers are used at the same time.

The reconstruction of digits (see Fig. 6.9) by a sparse autoencoder is a bit less sharp than reconstruction by an ordinary autoencoder (compare Fig. 6.6).

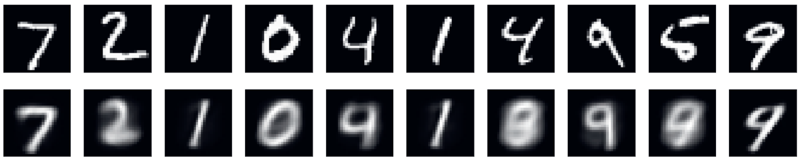


Figure 6.9: Reconstruction of MNIST data by a sparse autoencoder. The results are a bit less sharp than that of the ordinary autoencoder.

The scatterplot in Fig. 6.10 shows that only a few of the latent variables have non-zero values. The sparsity in the latent variables layer is achieved.

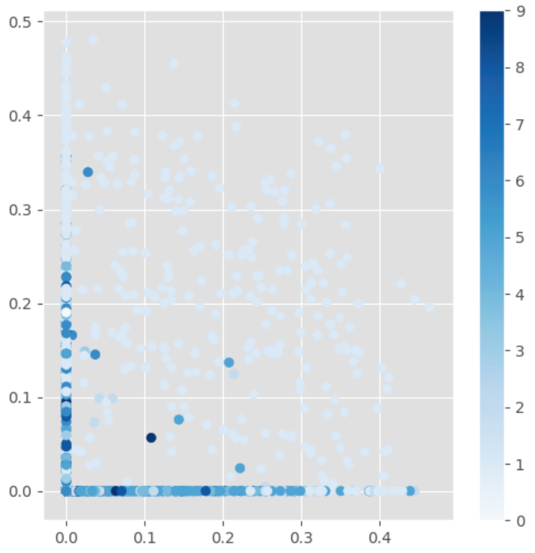


Figure 6.10: Scatterplot of latent representations in a sparse autoencoder. The two axes represent only two nodes in the latent layer. In contrast to the ordinary autoencoder (Fig. 6.7) is clear that inputs are mostly represented by either one or the other latent variable with a value that is nonnegative.

For most digits at least one of the first two nodes in the latent layer are zero (Fig. 6.10). We do not show the reconstruction of lines. The sparse autoencoder is failing in a similar manner as the ordinary autoencoder (Fig. 6.8).

6.2.4 Convolutional Autoencoder

The results for the dataset with multiple lines (Lines100) was shown for the ordinary autoencoder in Fig. 6.8. This dataset is also impossible to reconstruct for the variational and sparse encoder (not shown). The likely reason for this is that 2D data is inherently correlated. The (x, y) coordinates are dependent.

A convolutional autoencoder (Masci et al., 2011) is an autoencoder architecture that can operate on this type of data. It differs in its connectivity between the nodes in the autoencoder rather than in the loss function. The nodes are subdivided into layers. The layers are sparsely connected with each other, by convolutions. A stack of such layers has been given the name "deep neural network".

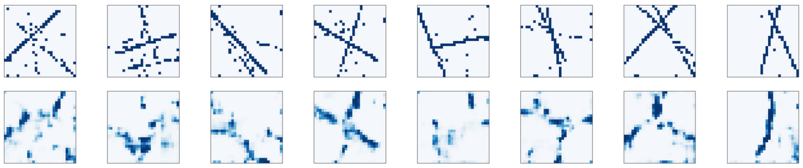


Figure 6.11: The reconstruction of lines (top row) starts to work for the convolutional autoencoder (results in bottom row). The reconstructions are blurry, but recognizable.

The convolutional autoencoder works on the 2D dataset if the lines are first mapped to a 2D grid. To use the same strategy in 3D, would mean that we have to create a voxel space. Rather than points like (x, y, z) we need to create a matrix of $LxMxN$, which becomes really large for increasing granularity. Although, the convolutional autoencoder shows that reconstructions are possible, we need something more sophisticated. In the next section we introduce autoencoders that are dedicated to point clouds.

6.3 Autoencoders on Point Clouds

Point cloud data has only recently been directly fed into deep neural networks. PointNet (Qi et al., 2017) is the first implementation of a deep network for segmentation and classification that directly operates on point clouds. In (Section 6.3.1) we describe an autoencoder from the literature which uses an architecture similar to PointNet as an encoder. The quality of this autoencoder will be assessed by visual inspection of the reconstruction quality (Section 6.3.2) and by a latent variable sweep from one representation to another (Section 6.3.3).

6.3.1 Earth Mover's Distance

The autoencoder (Achlioptas et al., 2018) architecture builds on PointNet. We refer to (Qi et al., 2017) for further details on PointNet. The autoencoder uses PointNet for the encoder. It accepts a point cloud with 2048 points (a 2048 x 3 matrix). It is constructed out of convolutional layers. The layers have kernel size 1 and an increasing number of features (representing the "neighborhood" of a point). The last layer is formed by a symmetric function, which is permutation invariant. The permutation invariance is required due to the exchangeability of points.

In a bit more detail, there are five convolutional layers, each is followed by a rectified linear unit (Nair and Hinton, 2010) and a batch-norm layer (Ioffe and Szegedy, 2015). The permutation invariant function takes a maximum (per feature) and forms a latent vector.

The decoder in this autoencoder is formed by three fully-connected layers with the first two followed by a ReLU and as output a 2048 x 3 matrix, the reconstructed point cloud.

One of the permutation-invariant objectives employed is the so-called earth mover's distance.

The earth mover's distance (EMD) is also known as the 1st Wasserstein distance. It can be considered an optimal transport problem. The concept has been first introduced in the 18th century (Monge, 1781). If there is an amount of sand and a pit where it has to go, how do we optimal transport the sand to the pit? The quantity that has to be minimized for this is called the earth mover's distance.

▼ **Definition 6.1 — *earth mover's distance***

The earth mover's distance (EMD) between $S_1, S_2 \in \mathscr{R}^3$ of equal size $|S_1| = |S_2|$ is defined as:

$$d_{EMD}(S_1, S_2) = \min_{\phi : S_1 \to S_2} \sum_{x \in S_1} ||x - \phi(x)||_2$$

with $\phi : S_1 \to S_2$ a bijection.

In the next section we apply this autoencoder using the earth mover's distance on 3D point clouds.

6.3.2 Reconstruction of 3D point clouds

The autoencoder properly reconstructs not just 2D lines, but complete point clouds. In Figure 6.12 the autoencoder uses the earth mover's distance as loss function to reconstruct the original cubes.

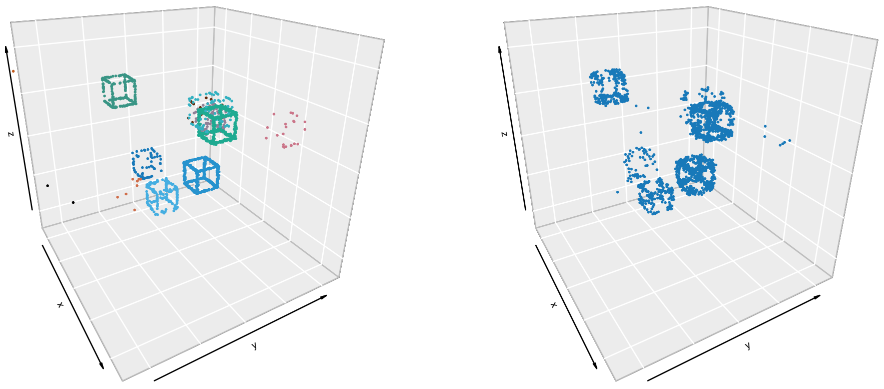
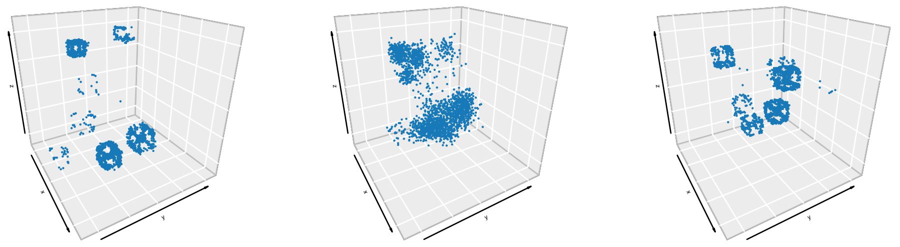


Figure 6.12: Reconstruction of a point cloud consisting of multiple cubes. The autoencoder uses the earth mover's distance. Left: the original point cloud. Right: the reconstructed point cloud. The reconstruction is not perfect. The cubes are recognizable though.

6.3.3 Quality of Latent Representation

To assess the quality of the latent representation we interpolate linearly between two latent variable representations that belong to two different cube configurations. Fig. 6.13 visualizes reconstructions of latent representations using the earth mover's distance. The results of the Chamfer distance are similar (not shown here).



(a) Reconstruction of latent representation corresponding to sample 1.

(b) Reconstruction of the latent representation linearly interpolated between latent representations of sample 1 and 2.

(c) Reconstruction of latent representation corresponding to sample 2.

Figure 6.13: Reconstruction of point clouds. This uses the earth mover's distance as a reconstruction loss. The representation in the center is generated by interpolating between two latent representations of actual samples. The reason that this is shown is to demonstrate that this latent representation is not generating geometric objects that are cubes. In contrast, it is a more general point cloud. This provides a hint that the autoencoder did not learn the concept of a cube. If it would, the intermediate representation more likely would have represented cubes at intermediate locations.

Remarkably, the objects at the interpolated steps do not resemble cubes [2]. The interpolation shows that the internal structure is not maintained. In between the two cube configurations, there is an unstructured point cloud, like Gaussian distributed blobs.

6.4 Semi-Autoencoders on Point Clouds

This section will address the limitations by the autoencoders and distance measures of the previous section. First we will explain limitation of the EMD with respect to uniformity (Section 6.4.1). We describe a first generalization of EMD called shifted earth mover's distance which addresses uniformity (Section 6.4.2). This is not able to cope with multiple objects for which we describe a second generalization called partitioning earth mover's distance (Section 6.4.3). This distance will then be used to create a semi-autoencoder (Section 6.4.4).

[2]A video can be seen at `https://bit.ly/2G2gyE2`

6.4.1 Limitations of the Earth Mover's Distance

The earth mover's distance defines a transportation map. Although the name optimal transport might suggest that there is a single optimal map, there are multiple maps possible. Depending on the implementation a particular map can be found. For example, if single grains of dirt are positioned at $x = 0, x = 1$ and need to be transported to $x = 2, x = 3$, it is possible to move one grain from $x = 0 \to 2$ and the other to $x = 1 \to 3$ or to move one grain from $x = 0 \to 3$ and the other $x = 1 \to 2$. The total distance traveled in both cases is 4. In the first case the moves are more uniform than in the second case.

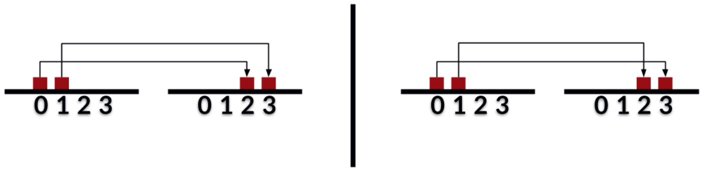
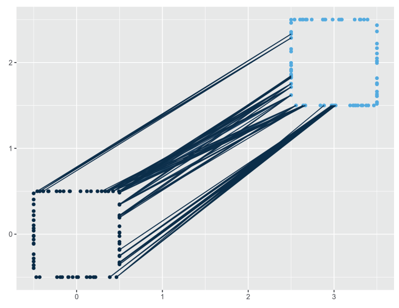


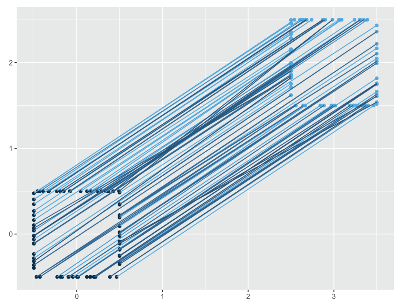
Figure 6.14: This figure shows that EMD is not unique. There are multiple maps possible and one map can be more "uniform" than another. Left: a map where each grain is moved with the same distance (of two). This is "uniform". We can achieve this with a global coordinate frame shift of two. Right: a map where one grain is moved over a larger distance (three versus one). This is not "uniform". Important to note, the total distance for both maps sums to four. The EMD can not distinguish between those cases!

The uniformity in a transportation map corresponds in the case of robotic vision with objects that are spatially translated. The transportation map that is not uniform does not correspond to a recognizable geometric operation.

In Fig. 6.15 we see at the left how we have the non-uniform transportation plan and on the right a uniform transportation plan.



(a) A non-uniform transportation plan. The square at the top-right is transported to the one at the origin. The points close to the origin are transported to the top-left of the square at the origin. Only the largest transport values are shown.



(b) A uniform transportation plan. A uniform transportation plan. The points in the square at the top-right are transported to the square at the origin in such a way that e.g. the points in a corner of the right square map to the points in the "same" corner of the other square. The map is uniform.

Figure 6.15: Earth mover's distance implementations. Both figures show "before" and "after" of the transportation map.

In Section 6.4.2 we describe an improvement on EMD which takes into account spatial translations. In Section 6.4.3 we describe a further generalization that takes into account multiple objects.

6.4.2 Shifted Earth Mover's Distance

The particular version of the earth mover's distance that is implemented is a shifted version of the EMD, the Shifted earth mover's distance. Suppose, in analogy with the original postulation of the problem by Mongei, that there is a small landslide such that the center of the source distribution becomes exactly on top of the center of the target distribution. The dirt now only has to be moved on a smaller scale.

The SEMD is calculated by shifting both pointsets such that they are centered around the origin. Note, that for datasets where the objects are already centered around the origin this will not make any difference. However, our dataset is definitely not centered around the origin, so this has a large effect (see again Fig. 6.15).

6.4.3 Partitioning Earth Mover's Distance

The SEMD is an improvement on EMD for single objects. However, for multiple objects the shift to the origin needs to be different for each object. PEMD partitions the space and couples each subset with its own shift operator. We visualize such a map in Fig. 6.16.

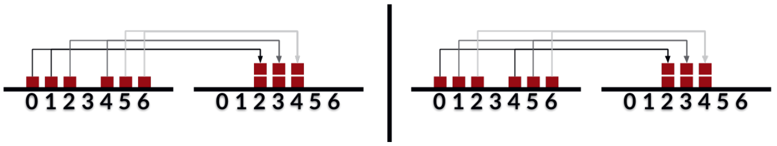
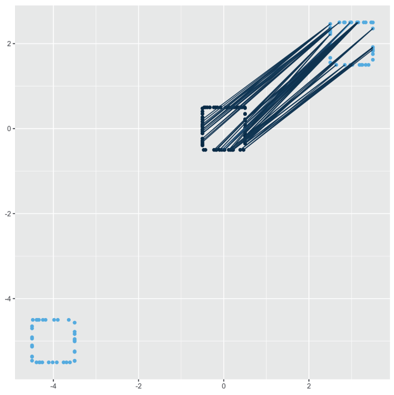
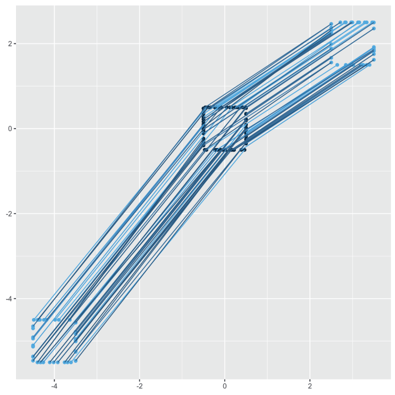


Figure 6.16: This figure demonstrates partitioning earth mover's distance (PEMD) with an example. Shown are two maps which are equivalent for EMD. Left: a possible SEMD map, the grains are moved by $\{2, 1, 1, -1, -1, -2\}$ steps (sum of absolute values is 8). Right: the PEMD map, the grains are moved by $\{2, 2, 2, -2, -2, -2\}$ steps (sum of absolute values is 12). The PEMD preserves the "structure" of the three grains in a cluster. The PEMD is larger than the SEMD.

For our robotic vision problems of identifying point clouds this can be visualized as well (see Fig. 6.17). Note that the object is a square and that there are multiple identical objects are to be recognized. This is the purpose of deploying the PEMD distance. It minimizes the distance to a "prototypical" object. In our autoencoder application, the result will be such a "prototype" object. The autoencoder will not reconstruct the complete input, it will construct a single copy of multiple identical objects that are present in the input. This is the property we use of this semi-autoencoder. We use the constructed output in Section 6.5 as input for our sampler.



(a) Shifted earth mover's distance visualization. The objects at $(-4, 5)$ and $(2, 2)$ are mapped to the object in the center. The (uniform) mapping of SEMD does not distinguish between the squares. It is not preserving the structure of the individual squares.



(b) Partitioning earth mover's distance visualization. The objects at $(-4, 5)$ and $(2, 2)$ are mapped to the object in the center. The mapping maps both squares on top of the center square. It preserves the structure of the individual squares.

Figure 6.17: Earth mover's distance implementation on three objects. Left: SEMD. Right: PEMD.

The PEMD needs to find partitions. This is actually a clustering problem on its own. Given a pointset and a repeated structure, can we divide this pointset into that structure and the number of times this structure occurs?

We solve this by the following optimization procedure. We search for local modes using a conventional algorithm, in our case mean-shift. After we have found the local modes, we use the means of the data points we found to translate the objects in both point clouds to the origin. We then perform EMD on the entire (normalized) dataset.

In Fig. 6.17 the difference can be seen in the transportation plan between a global shift (left) and shifts per partition (right). The shifts per partition shows a transportation map that preserves the local structure.

6.4.4 Semi-autoencoder

There are examples of generalizations of an autoencoder where the dimensions of the input and the output does not have to be identical (Zhang et al., 2017). However, the discrepancy between input and output can purely rise from the distance function used as well. If the distance function is truly translation invariant, the output gives rise to the object searched for. It is not a matter of reconstructing one of the input objects. A proper autoencoder would reconstruct a single object using information from all those duplicate objects in the input. This map can be seen as a generalized reconstruction objective. Alternatively, it can be seen as a general encoder-decoder network where we search for transformations of the input rather than the identity operation (Worrall et al., 2017).

The semi-autoencoder can be seen as a system that separates the object's shape (the "what") from the object's position (the "where") tailored to a situation where there are many identical objects. Separation between object shape and position through separate pathways seems ubiquitous in the (mammalian) brain (Ungerleider and Haxby, 1994; Rauschecker and Tian, 2000). Even more relevant to our architecture, there is brain-imaging evidence that the pathways for object identity and numerosity (the number of objects) are separated (Izard et al., 2008). Moreover, the ability to represent numerosity seems just as ubiquitous as the what-where pathways and can for example be found in mosquitofish (Agrillo et al., 2011) and honeybees (Bortot et al., 2019).

In this context, the semi-autoencoder fulfills the role of object identification. Given an object there is another pathway that is able to reason with these objects. In our case we will use a triadic sampler to perform class assignments. When points are assigned to multiple objects we have indirectly also represented counting those objects. In other words we have a second pipeline to establish numerosity. Note, that in contrast with subitizing autoencoders (Wever and Runia, 2018; Pecyna et al., 2019) our objective is not counting itself but clustering: the proper assignment of 3D points to multiple 3D objects.

6.5 Results

In Section 6.5.1 we describe a few implementation details for the autoencoder. In Section 6.5.2 we analyze the performance of the classifier that uses the results of the autoencoder for the task of assigning points to cubes in point clouds.

6.5.1 Implementation

The autoencoder using the partitioning earth mover's distance[3] is based on an autoencoder implementation using PointNet (Achlioptas et al., 2018). It uses Tensorflow and the metrics and their gradients have been implemented for the CPU (python3) and for the GPU (CUDA). The implementation for the GPU aims to improve the speed of the operations.

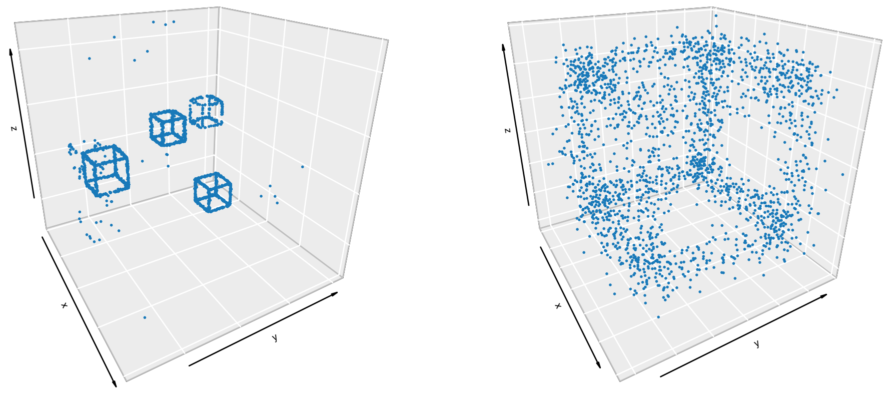


Figure 6.18: Left: the input, a sample with multiple 3D cube objects. Right: a reconstructed cube. The autoencoder uses partitioning earth mover's distance as a metric. The reconstruction creates a single object. Each cube in the original input contributes to the representation in the output. The output reflects the invariance in number of objects and their locations.

In Fig. 6.18 one of the reconstructed samples is visualized. The input, the set of cubes, is at the left. The output, the single cube at the origin, is at the right. This is a to be expected result from the reconstruction process. The loss function does not take into account the actual position of the cubes: it is on purpose invariant to these positions. This means that the generation process yielding a unity cube will have a very low loss associated to it.

6.5.2 Clustering Performance

The reconstruction of the autoencoder is used as a reference object for a triadic MCMC sampler (Chapter 5). The MCMC sampler compares the reference object with the current sample. Each point is compared with its nearest neighbor in the reference set and its match is defined through a normal distribution. The datasets are subsampled to 200 points.

More specific, it is common to compare two objects using pairwise Euclidean distances between (correspondence) points (Boutin and Kemper, 2004). We will use a Dirichlet Process with a multivariate normal distribution as base distribution, H, centered at the origin and with scalar noise ($\sigma = 1$). During the inference process the sampler will generate 3D locations, μ_i, for the hypothesized objects and we will calculate the difference between the

[3]Implementation at `https://github.com/mrquincle/latent_3d_points`.

hypothesized object with the reference object using the Euclidean kernel. In our implementation the objects do not have to have the same number of points. We sum the distances of the closest point (also with respect to the Euclidean distance). This means that the reference object can be much denser than the hypothesized object.

We can describe the Dirichlet process in the usual manner:

$$\begin{aligned} G &\sim DP(\alpha, H), \\ \mu_i \mid G &\overset{iid}{\sim} G, \\ w_i \mid \mu_i, r &\sim F(w; \mu, r). \end{aligned} \tag{6.4}$$

The likelihood function has a bit of a complicated structure:

$$F(w_i|\mu, r) = D(w_i, g(w_i, f(r, \mu))). \tag{6.5}$$

The reference point cloud r is undergoing an operation through $f(r, \mu)$, in this case a simple shift $f(r_j, \mu) = r_j - \mu$ for each j in the point cloud r. The function g is nonlinear and finds the point closest to w_i in r, which we define as r'. Then the distance is calculated between w_i and r' through D, the Euclidean distance.

The results of the triadic sampler can be found in Table 6.1.

Table 6.1: The purity, rand index, and adjusted rand index establishing the quality of the clustering method for line estimation (Chapter 5) and cube estimation (this chapter). The more complex dataset results in to be expected lower performance levels, but a significant number of assignments are correct. The results with the other types of samplers are not repeated for the cube dataset considering that they underperformed the triadic sampler already in the line estimation task.

Dataset	Purity	Rand Index	Adjusted Rand Index
Line estimation (Chapter 5)	0.86405	0.87188	0.71067
Cube estimation	0.8367	0.8359	0.6524

The spread over the samples is displayed in Figure 6.19 as a violin plot. Note that there might be still room for improvement. For some data samples an adjusted rand index of 0 corresponds to chance. Yet, it does not necessarily mean that the sampler can be improved. If there are two cubes generated at exactly the same location the "correct" cube becomes unidentifiable.

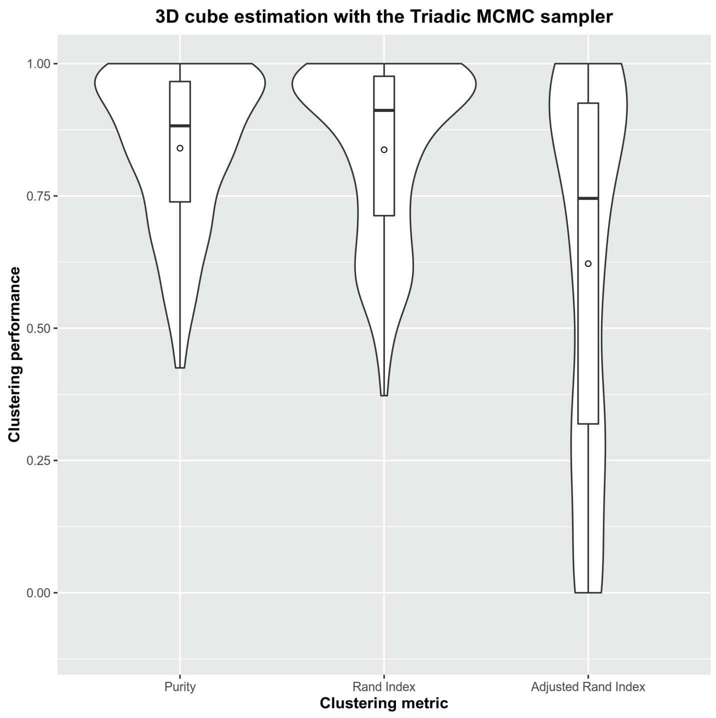


Figure 6.19: Performance of the triadic sampler on 3D cubes.

Figure 6.20 shows an assignment for a particular sample from the dataset. The cubes are properly identified as separate entities. There are also a few points that are accidentally assigned to those cubes as well.

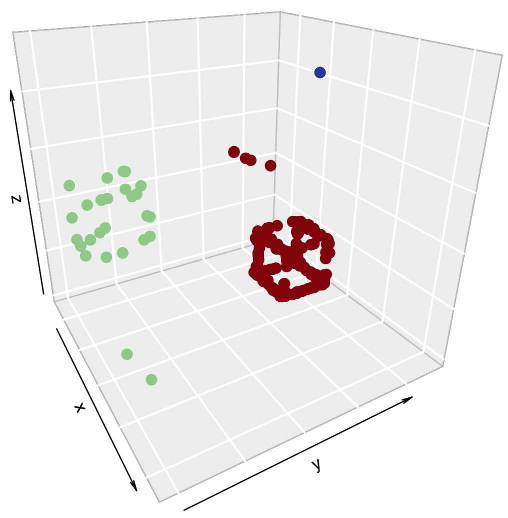


Figure 6.20: One of the samples as classified by the triadic sampler. The assignments of points to cubes is indicated by colors. It is visualized how some of the points are misclassified. In this case this happens mainly for cubes with only a few points.

6.6 Chapter Conclusions

The previous chapters went to great lengths to use Bayesian inference methods to given prior and likelihood optimally infer lines, line segments, squares, and other primitive geometrical objects. In this chapter we deployed deep learning methods to be able to reason about less primitive objects: cubes. A naive application of state-of-the-art deep learning methods for point clouds is not sufficient. Compared to dataset in the literature the objects in our dataset are shifted, and importantly, there are multiple objects in a single frame. The neural network needs to learn multiple objects at once.

The chapter introduced two new metrics for autoencoders based on earth mover's distance or Wasserstein. First, a SEMD or Shifted Wasserstein metric that can be used on dataset where objects are not centered at the origin. Second, a PEMD or Partitioning Wasserstein metric that can be used for datasets where individual frames do have multiple objects.

The results show that it is indeed possible to create a variant of an autoencoder that learns to reconstruct a particular object when there are multiple copies in its input. It is a semi-autoencoder, not an exact autoencoder. The purpose is not to exactly regenerate the input, but to represent the duplicate objects as one single object. It uses the same encoding-decoding structure as an autoencoder.

The model in this chapter is able to:

- Perform inference on multiple objects by using a partitioning earth mover's distance. Inference over multiple objects simultaneously is not part of recent optimal transport literature (Alvarez-Melis et al., 2018, 2019). The ability to perform inference over multiple objects is neither part of translation-aware work such as transforming autoencoders (Hinton et al., 2011) and capsule networks (Hinton et al., 2018).
- Perform inference on multiple objects without defining the number beforehand. This goes beyond e.g. the work on so-called barycenters (Forrow et al., 2018, 2019).

This chapter demonstrated how to perform inference on more complex volumetric objects. It showed how naive application of autoencoders - even autoencoders especially designed for point clouds - fails for inference of multiple objects.

New metrics and a new type of autoencoder is developed to take this type of structure into account. Such an autoencoder can then be used for a prior for a Bayesian model. This model can be sampled with the techniques of the previous chapters.

This chapter answers our third research question.

RQ 3 How can we recognize more general 3D objects?

First we use modern deep learning techniques to create data-driven priors. These priors can be very complex, learned - over many iterations of the data - by an autoencoders with many layers. Second, given these data-driven priors we perform inference using a nonparametric Bayesian model to detect general 3D objects. This bridges the field of deep learning with that of nonparametric Bayesian methods.