University of Exeter – College of Engineering, Mathematics and Physical Sciences

Manifold learning for explaining the behaviour of recurrent neural networks

01-05-2019

Abstract

In a world where AI is becoming a prominent tool in our everyday lives, both for trivial tasks and in more critical settings, it is crucial to develop ways to understand how the machine makes every decision. In particular, widely used tools are Echo State Networks (ESNs), which are used in making predictions about temporal-varying data and sequences. Ceni's [1] research focuses on developing a method to describe the behaviour of a given ESN by creating an Excitable Network Attractor (ENA) that describes its internal dynamics. The main steps in the development of the network involve finding the location of the attractors and the relative excitability thresholds. The latter is performed in a quite simplistic manner in Ceni's paper, affecting the accuracy of the remainder of the project. Finding ways to improve on such calculation will be beneficial to every further result in Ceni's project.

To do so, we focused on improving the intermediate steps between finding the attractors and calculating the thresholds. A more compact representation for the effect of inputs on the network has been found with non-linear manifold learning techniques and the effect of new inputs has been simulated using advanced sampling methods. Finally, by means of a dimensionality-increasing neural network, we set the basis for a more accurate calculation of the excitability thresholds using Ceni's approach.

Once the above methods have been developed and tested, the result of each method has been compared with Ceni's approach for each step. According to such tests, the joint application of all the new methods introduced an improvement of about one order of magnitude, making the contribution of this paper on Ceni's work significant as it brings a great improvement on the quality of the estimation.

I certify that all material in this dissertation which is not my own work has been identified.

Summary

1 Introduction: background and aim of the project

1.1 Overview and project statement

As AI and machine learning become increasingly important in our everyday lives, it is time to start asking ourselves questions about these methods. A central task in AI is to make predictions about time-varying data [2]; this is done using Recurrent Neural Networks (RNNs) [3] and Echo State Networks (ESN) [4]. While these tools work reasonably well in a large set of tasks, they also share an important weakness: the way they elaborate their input is not clear at first glance [5]. This problem may seem trivial, but when applying any tool in situations where moral choices are involved (the classical rhetorical question: should a self-driving car kill its owner to save more lives in an accident?) or in a scientific environment (when using RNNs and ESNs as research tools), it is no longer sufficient to know that such tools "work". We also need to know *how* they operate to develop accountability of the decisions taken by the machine.
The wider scope of the project will therefore be to develop a way to explain how these tools work. In practice, we try to find a way to transform instances of ESNs, whose inner workings are not fully understood, into something of which we have a higher degree of understanding: Excitable Network Attractors (ENAs) [6].

The aim of this project is to improve on elements of the methodology adopted by Ceni et al. [1] to establish the excitability thresholds of the ENA corresponding to a given ESN.
To do so, we will create and apply better ways to perform different tasks in Ceni's research. Since the project is not aiming to revolutionise the whole of Ceni's study, we will work on the assumption that the first part of the project has already been done (namely: we have a trained ESN and the location of the attractors in the ENA is known, [1], Sec. 2.2), and we will focus on the second part.
The focus of the project will therefore be to devise a way to improve the currently implemented estimation of excitability thresholds found above, and also proving that such newfound methods yield better results than those already in place. This would allow Ceni's research to reach a new standard of accuracy in the estimation of the thresholds, improving all the consequent results of the project.
Following the steps outlined in Ceni's work, Section 4, we can give a short outline of the procedure used here to provide a more accurate estimation of the thresholds:

1. Firstly, we will need to analyse how each input affects the dynamics of the machine; this is done by studying the difference in the state of the machine between before and after feeding a series of inputs to the ESN. Each one of these differences will be stored in a vector called *Pulse Different Vector* (PDV) and it will be represented as a point in the proximity of each attractor. As theorised by Ceni, these points lay on lower-dimensional manifold $\mathcal{G}$; our first step will therefore involve the modelling of such manifold.
2. Then, we need to sample new points that lay on the same low-dimensional surface as the original data to obtain a much larger dataset without the need to use the ESN. Sampling new points from the found manifold will be the second step of the project.
3. Finally, it is necessary to invert the dimensionality reduction process to create new high-dimensional points coherent with the original data. By running the ESN and studying the convergence of all the points in the manifold we can now estimate the area of influence of each attractor and hence infer the excitability thresholds.

1.2 Project steps

The work proposed by Ceni et al. [1] defines the general structure of the procedure to be used in order to establish the excitability thresholds. Nonetheless, it has some major pitfalls that need to be addressed: (i) by using PCA to model the manifold they are only able to model a linear, hyper-planar, subspace, which is not a good representation of the actual distribution of points in the high-dimensional space [7, 8]. This implies that (ii) also the next steps of the algorithms will be affected by the non-optimal representation of the manifold and therefore (iii) making the calculation of the excitability thresholds inaccurate.
The improvement proposed here is significant because it gives us the tools to perform a better estimation of the thresholds. (i) By using non-linear manifold learning methods we will be able to more accurately model the underlying manifold [9]; (ii) this will allow us to sample points with an improved accuracy. (iii) The development of the tools hereby proposed will therefore be central in estimating the excitability thresholds of the network with increased precision.
From the above paragraphs we can infer what the main steps of the project will be:

1. First we need to establish d, the number of intrinsic dimensions of $\mathcal{G}$.
2. It might be beneficial to cluster the dataset if the manifold will result to be fragmented.
3. Then it is necessary to determine the structure of the manifold on which the points of the local PDVs lay. In Ceni et al. [1] this is approximated by an hyper-plane while we aim to find a more complex manifold.

4. From the manifold, there is the need to sample new points. In this paper are proposed both a geometrical and a statistical solution.
5. Once new points have been sampled, it is necessary to find their pre-images in order to use the new points as starting points for the next iterations.
6. It is also necessary to find a way to embed new points in an already built manifold to reduce the computational cost when new data are added to the system.

2 Literature review overview

2.1 Dimensionality reduction

The first step requires us to model the low-dimensional manifold. The problem can be summarised as finding a representation of the points in a lower-dimensional space while retaining most of the information content of the original data [8].
We denote with D the original dimensionality and d the target dimensionality, we also call $\mathbf{X}$ the high-dimensional dataset and $\mathbf{Y}$ its low-dimensional counterpart.
To solve this problem we propose five different methods, each with its own advantages and disadvantages: Principal Components Analysis (PCA)[8], Kernel PCA (KPCA) [10, 11], Isomap [12], Locally Linear Embedding (LLE) [13] and Autoencoders [8, 14, 15].
Some of the methods proposed share a common weakness: they do not work well when the original dataset has holes or is fragmented (divisible in clusters) [8]. For this reason, before applying such methods it might be necessary to cluster the data. First we need to determine the right number of clusters, this is done using the Davies Bouldin index, that aims to minimise the ratio between intra-cluster and inter-cluster distances [16], and the Elbow method, that looks for a discontinuity in the inter-cluster mean sum of squares plot [17].
To cluster the data we make use of Spectral Clustering as, by being a graph-based method, it is able to model a larger variety of structures, not necessarily Gaussian-like, but also spheres, moons and other non-trivial shapes [18]. Since the PDVs dataset is usually rather small, the high computational cost is not a concern here.

2.2 Out-of-Sample extension (OoSE)

As dimensionality reduction is usually an expensive operation, it is also necessary to find a way to embed new points into the manifold without recomputing it for all the original points.
Ideally this method should not depend on the way the manifold is constructed to allow more flexibility in the choice of the dimensionality reduction method.
Here we propose two methods: (i) Generalised Out-of-Sample Extension, that extracts a mapping by studying the change in geometry between the high and low dimensional points. The main advantage of this method is its independence from the manifold learning method used [19]; (ii) in case an Autoencoder will be used for the reduction, the out-of-sample extension problem is solved automatically by passing a new point to the network and reading the low-dimensional point off its middle layer [14]. Alternatively, a neural network can be trained to perform OoSE [20].

2.3 Sampling

The task can be formalised as: given a set of points on a manifold, find a new point (or set of points) on the given manifold, in the proximity of the original points and roughly following the same distribution.
We hereby propose two solutions, developed by us specifically to solve this problem:

1. Geometrical approach - Nearest Neighbours Interpolation: a graph-based method that relies on stochastic interpolation within a k nearest neighbourhood of a randomly selected point.
2. Statistical approach - Distribution Fitting: this method involves finding a known distribution that can accurately describe the given data and its parameters, once this has been done (if at all possible) sampling new points is just a matter of sampling new points from a distribution and is therefore trivial [21].

– A note on Autoencoders: in the case where a variational Autoencoder is used to reduce the dimensionality, sampling becomes a trivial problem as it can be reduced to sampling from a standard Gaussian distribution with the found parameters.

2.4 Pre-images problem

Once a point has been sampled from the manifold, it is necessary to reconstruct its high-dimensional representation in order to use it as an initial point for the iteration described in Section 1.1.
In order to do so we will need to learn a function from $\mathbf{Y}$ to $\mathbf{X}$ that produces high-dimensional points that are consistent with the rest of the points in $\mathbf{X}$.
The easiest way to solve this problem is by using an Autoencoder. This can be done without further effort when an Autoencoder is used to reduce the dimensionality of the data, but a neural network can also be trained specifically for this task if another method for manifold learning will be chosen.

3 Structure of the remainder of the report

The rest of the document will be structured as follows:

1. In Section 4 we will go over the main design choices and tests structures used in the development of the project, as well as giving a general blueprint of the tools that we are going to implement.
2. In Section 5 an in-depth analysis of the development process will be given, with specifics of the code and algorithms used.
3. In Section 6 the results of the tests designed in Section 4 will be be given, as well as a first analysis of such results.
4. In Section 7 we will apply the developed tools to the original problem and evaluate the performances, both overall and compared to the methods previously in place, aiming to improve in every aspect where a new method is introduced.
5. Finally, in Section 8 we will give a conclusion derived from the tests of Sec. 6 and Sec. 7. From these results we will also try to elaborate on what some possible future steps to further improve the accuracy of the model might be.

4 Design

4.1 Technology choice

Throughout the project we will be using Python, with the support of Numpy, Scikit-Learn and Keras.
The choice of the programming language fell on Python due to its popularity in the field of machine learning in general, and especially in the development of neural networks. The popularity of the language is imputable to its simplicity and readability for the user and its relative ease in prototyping when compared to other leading languages, such as Java or C. Moreover, throughout the years, Python has accumulated an incredibly vast number of libraries for a large set of applications and, in particular, it can boast some of the most advanced libraries for data analysis, scientific computing and machine learning in general.
One of the most used libraries used to expand the ability of vanilla Python to represent data structures is Numpy, which has a native support for vectors and matrices and their respective operations. Numpy also has some basic statistical tools and linear algebraic tools that can be useful in a variety of applications. Finally, it also speeds up the computations significantly by relying on a C++ compilation and implementation of the basic tools.
Scikit-Learn is one of the main tools used in data mining and data analysis in Python, providing support for classification, regression, clustering, dimensionality reduction and other pre-processing operations. By making use of Numpy in its calculations, it is also a great speed-up compared to the same methods implemented in Python from scratch.
Finally, Keras is a high-level library that wraps around TensorFlow and is used in prototyping neural networks. Keras supports a variety of standards neural networks building blocks, such as recurrent layers and feed-forward layers, but it also allows the user to implement custom layers.

4.2 Clustering

As reported in Section 2.1, some of the methods used to reduce the dimensionality of the data perform generally worse if the data are fragmented. In order to reduce the fragmentation of our dataset we are therefore going to determine if the data are divisible in clusters and, in that case, we are going to divide it and apply all the consequent methods on each cluster separately. The need of partitioning, and the correct number of clusters, will be evaluated using the Davies-Bouldin index [16] and the Elbow method [17].
Specifically, the DB index aims to find the minimum of

$$\mathrm{DB}(\mathcal{C}) = \frac{1}{k}\sum_{i=1}^{k} \max_{j \leq k, j \neq i} D_{ij}, \quad k = |\mathcal{C}|$$

where D_{ij} is the inter-to-intra cluster distance ratio for the i^{th} and j^{th} clusters. In particular

$$D_{ij} = \frac{(\bar{d}_i + \bar{d}_j)}{d_{ij}}$$

A proposed pseudo-code for calculating such index can be found in Algorithm 1. Let us recall that, when using the DB index, the aim is to find a value of k for which the index is minimum.

The Elbow method, instead, aims to find a discontinuity in the plot of the mean within-cluster sum of squares (WSS). The mean WSS measures the compactness of the clustering and we want it to be as small as possible while keeping a relatively small number of clusters. A method for calculating the WSS can be found in Algorithm 2
These two methods are relatively straightforward to describe and implement and do not need additional testing procedures.

For both algorithms the application will proceed as follows:

1. *Compute clustering algorithm (e.g. k-means) for different values of k. For instance, by varying k from 1 to 30.*
2. *For each k, calculate the DBI and the WSS.*
3. *Look for a minimum of the DBIs and for a discontinuity in the plot of the WSSs to find the correct number of clusters. Ideally the two should coincide.*

To check that the methods are implemented correctly, an artificial dataset made of three isotropic Gaussian blobs in 3 dimensions with standard deviation equal to 1 will be used, we expect both method to agree on three as the correct number of partitions.

Algorithm 1 Davies-Bouldin index calculation

Input: k, number of clusters; P, cluster assignment; X, original dataset.
Output: the DB index of the given cluster assignment.

$total\_distances \leftarrow \Lambda$

for i *from* 0 *to* k **do**
  $distances \leftarrow \Lambda$

  for j *from* 0 *to* k **do**
    if $i == j$ **then**
      skip
    end if

    $c_i \leftarrow \text{mean}(cluster\_i)$
    $c_j \leftarrow \text{mean}(cluster\_j)$
    $d_{ij} \leftarrow \sqrt{(c_i - c_j)^2}$

    $\overline{d}_i \leftarrow$ average distance between every point in cluster i and c_i
    $\overline{d}_j \leftarrow$ average distance between every point in cluster j and c_j

    $D_{ij} \leftarrow (\overline{d}_j + \overline{d}_i)/d_{ij}$
    $distances \leftarrow distances + D_{ij}$
  end for

  $total\_distances \leftarrow total\_distances + \max(distances)$

end for

return $\text{sum}(total\_distances)/k$

Algorithm 2 Elbow method

Input: k, number of clusters; P, cluster assignment; X, original dataset.
Output: the average sum of squares of all the points in each cluster, needed to plot the elbow method.

$sum\_of\_squares \leftarrow \Lambda$

for i *from* 0 *to* k **do**
  $centre\_i = \text{mean}(cluster\_i)$
  $sum\_of\_squares.append(\text{sum}((\text{points\_cluster\_i} - centre\_i)^2))$
end for

return avg($sum\_of\_squares$)

4.3 Dimensionality reduction

4.3.1 Finding the number of dimensions

The first step in the dimensionality reduction process is to determine the number of dimensions needed to represent the data with a given accuracy.
This will be done using residual variance with PCA. The use of PCA here is justified as it is the "worst", or least flexible, method used in dimensionality reduction. The intuition behind this choice is that, if even PCA can describe the data while preserving a given amount of information at a certain dimensionality, it is likely that the other methods will be able to achieve similar or better results in the same setting [22, 12].
Example: we can set the target residual variance to 10%, we then reduce the dimensionality of the data for $d \in [1..100]$ and measure the residual variance in each case. We can then pick the correct number of dimensions by taking the lowest d for which the residual variance is $\leq 10\%$.

4.3.2 Reducing the dimensionality

As mentioned in Section 2.1, the following methods will be considered for testing: PCA, Kernel PCA, Isomap, LLE and Autoencoders (standard and variational). Each of these methods has its advantages and disadvantages, and therefore the choice of the method to use is not trivial and more experimentation will be required.
A brief theoretical comparison follows [8, 10, 11, 12, 13, 14, 15]:

Method Name	Linear	Data preserved	Advantages	Disadvantages
PCA	Yes	Variance and large pairwise distances.	Simplicity of method and implementation.	Linearity, type of distance preserved.
Kernel PCA	No	Variance and large pairwise distances in kernel space.	Simplicity of method and implementation.	Type of distance preserved.
Isomap	No	Geodesic distances and short pairwise distances.	Type of distance preserved.	Computational cost, problems if data fragmented.
LLE	No	Local linearity of patches of the manifold.	Only one hyper-parameter and tractable problem.	Manifold might not be locally linear, problems if data fragmented.
Autoencoders	No	Finds encoding and decoding functions for which $d(e(\cdot))$ is as close as possible to the identity function.	Automatically solves OoSE. Variational Autoencoders solve sampling as well.	Large number of hyper-parameters, expensive training process.

Table 1: A short comparison between the proposed dimensionality reduction methods

It is also important to remind the reader of the structure of Autoencoders to better understand how they might be useful in the scope of the project: an Autoencoder is a (usually) symmetric neural network with an odd number of layers. An Autoencoder can be divided in two parts: from the input layer to the central layer (encoder), and from the central layer to the output layer (decoder), let us call these layers i, m and o.
The encoder has input dimension equal to D and output dimension (through m) equal to d, the decoder has input dimensions d (from m) and output dimension D. Because of this structure, the decoder is capable of decreasing the dimensionality of the input points and the decoder is capable of increasing the dimensionality of the produced low-dimensional points [8].
A Variational Autoencoder differs from the standard implementation as, instead of having a single middle layer, it

produces standard deviation and mean vectors, then samples a point with the found parameters from a Gaussian distribution and proceeds with the decoding part as before [15].
Using Autoencoders the OoSE problem is automatically solved by passing the new points to the trained network and using VAEs the sampling problem is also trivial as it is just a matter of sampling from a Gaussian distribution. Structures for both a standard Autoencoder and a Variational Autoencoder can be found in Fig. 1.

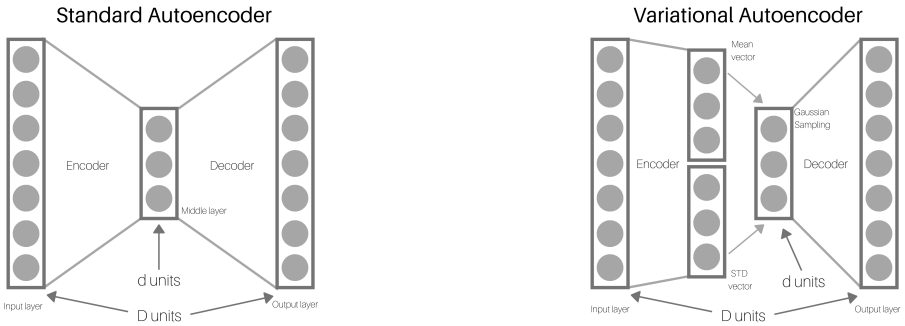


Figure 1: The basic structure of Autoencoders

4.3.3 General testing procedure

The implementation of all dimensionality reduction methods will follow as closely as possible the description given in the literature review section of the report. What is left to be discussed is how the performances of all the proposed methods are going to be evaluated to be ultimately able to choose a subset of algorithms better suited for the task at hand.
The first intuition in this area is that we want our dimensionality reduction to preserve as much information as possible rather than maintaining the topological structure of the data. In practice, we want little to no difference in the application of machine learning algorithms before and after the reduction. A proposed solution for this intuition is to pick a simple algorithm, such as a K-Nearest-Neighbours Classifier (KNNC), choose a set of datasets with known classification (Table 1), and use KNNC to classify the datasets before and after the reduction (performed with all the methods above). By comparing the accuracy of the classifications, we obtain an empirical measure of the quality of the reduction by estimating how much information has been lost in the process.
From the first intuition it appears clear that we now have the need to find adequate datasets on which to apply the method described above. We can start by noticing some properties of the final dataset on which we are going to apply the dimensionality reduction (the dataset of PDVs produced by the ESN) and look for other datasets (with known classification) with similar characteristics. Firstly, the dataset has a high starting dimensionality (500 features); secondly, we want to reduce it to a very low number of dimensions; and finally, we want data from the real world, avoiding synthetic datasets. The table below shows the datasets chosen according the these criteria. Notice that "Blobs" is a very simple synthetic dataset that we are going to use as a control set, rather than a proper benchmark.
In short, the method used to evaluate the performance can be summarised as follows:

1. *Pick a dataset.*
2. *Use KNNC to classify it and evaluate the accuracy (x%). We will use $k=3$ in our tests.*
3. *Reduce the dimensionality of the dataset.*
4. *Apply KNNC again on the low-dimensional dataset and evaluate the accuracy (y%).*
5. *The loss in accuracy is simply $(x-y)\%$, where generally $x \geq y$. Notice that y can be $> x$, meaning that, reducing the dimensionality, we were also able to improve the performance of the chosen classifier in the process.*

According to the above considerations the following datasets and target dimensions have been chosen for the tests:

Name of the dataset	# samples	# classes	D	d
Olivetti faces	400	40	4096	16
Digits	1800	10	64	8
Blobs	2000	5	200	2
Parkinson	756	2	753	1
Madelon	2598	2	500	3
HTRU	17897	2	8	2

Table 2: Datasets chosen to test the performances of the proposed dimensionality reduction methods

The "Olivetti faces" dataset contains a set of face images. There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position [23]. This dataset was chosen because of its high number of dimensions to put the dimensionality reduction methods under stress.
The "Digits" dataset is the MNIST digits dataset and contains greyscale images of handwritten digits, their size have been normalised and centred [24].
The "Blobs" dataset is made of five isotropic Gaussian blobs with standard deviation equal to 1 enclosed in a 1×1 box.
The "Parkinson" dataset consists of information about speech features gathered from 188 patients with Parkinson Disease (107 men and 81 women) with ages ranging from 33 to 87 at the Department of Neurology in Cerrahpasa, Faculty of Medicine, Istanbul University. The control group consists of 64 healthy individuals with ages varying between 41 and 82 [25].
The "Madelon" dataset is an artificial dataset containing data points grouped in 32 clusters placed on the vertices of a five-dimensional hypercube and randomly labelled +1 or -1. The five dimensions constitute 5 informative features. 15 linear combinations of those features were added to form a set of 20 (redundant) informative features. The dataset is highly non-linear [26].
The "HTRU" dataset describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey [27]. This dataset was chosen because of its high number of samples to put the dimensionality reduction methods under stress.

4.3.4 Autoencoders

A separate note is required for the design of Autoencoders as their structure greatly impacts the performance of the network and therefore it needs to be evaluated carefully [8].
Since all the hyper-parameters are not known at the design stage, we are just going to state which hyper-parameters we will make experiments on, and how these experiments will be conducted for each parameter.
From an analysis of Figure 1, and keeping in mind the training process of neural networks, we can infer that the main hyper-parameters of the network will be:

1. The number of hidden layers
2. The size of the batches of data used in the training process

The above parameters will be tested against the datasets in Table 2 with the method proposed above. The testing will focus on one parameter at a time, leaving the other unchanged.
The number of hidden layers will be varied from 3 to 7 (3, 5, 7), while keeping the number of hidden units fixed to $[3D/2, D/2^{i-1}, d, D/2^{i-1}, 3D/2]$, where i represents the i^{th} hidden layer, and the batch size fixed to 128 (e.g. if the number of hidden layers is 7, $D = 128$ and $d = 4$, the number of units for each layer would be [192,128,64,4,64,128,192].
The batch size will be changed from 32 to 256 in powers of 2, while keeping the number of hidden layers to 5 and the number of hidden units to $[3D/2, D, d, D, 3D/2]$.
Once all the parameters have been tested individually we will check that the combination of the best parameter in each test is indeed better, or similar, to each singular best.

4.4 Out-of-sample extension

As for Section 2.2, we propose three methods to embed new points in an already-built manifold. In case an Autoencoder was used to reduce the dimensionality of the data, recalling that an Autoencoder can be split into two parts, an encoding part, that reduces the dimensionality of the data, and a decoding part, inverting the reduction process, we can simply use its encoding part to reduce the dimensionality of unseen data-points [8]. As

the network will have already been built and trained at this stage, no additional work is required. For all other cases we can use the Generalised Out-of-Sample Extensions method (GOoSE) [19], which does not depend on the original dimensionality reduction method used to produce the manifold and can therefore be applied to all other methods.
A third option, viable in the case where the dimensionality reduction method is not an Autoencoder and if GOoSE should fail, would involve training a neural network with D units in its input layer and d units in its output layer on the high and low-dimensional representations of the same points to make it learn the function used to reduce the dimensionality and then apply it to unseen points [20].
While the former method is straightforward and its design has already been discussed in the section above, the latter requires a general code structure to be laid down and to be later followed for a practical implementation.

In all cases we will want to assess the performance; an experiment process has been devised to obtain a numerical value describing the accuracy of the method:

1. *For each dataset from table 2, divide it into a train set and a test set (with ratio 80:20).*
2. *Reduce the dimensionality of the whole dataset.*
3. *Train the methods on the train portion of both the high-dimensional part and the low-dimensional part.*
4. *Reduce the dimensionality of the test set.*
5. *Compare the newly produced low-dimensional points with the point produced in point 2.*
6. *Using a measure of distance (e.g. MSE) we can now evaluate and compare the accuracy of each method.*

Algorithm 3 Generalised Out-of-Sample Extension

Require: $x \in \mathbb{R}^D, \mathbf{X} \in \mathbb{R}^D, \mathbf{Y} \in \mathbb{R}^d, k \ll |\mathbf{X}|$

$\mathbf{X} \leftarrow \mathbf{X} + x$ # the point is added to the dataset
$\mathbf{X}_\mathcal{N} \leftarrow k$ nearest neighbours of x
$\mathbf{Y}_\mathcal{N} \leftarrow$ low dimensional representation of $X_\mathcal{N}$
$\mathbf{Z}_\mathcal{N} \leftarrow PCA(\mathbf{X}_\mathcal{N})$ # here the number of dimensions of PCA is the same as that of Y
$\mathbf{U}\Lambda\mathbf{V} \leftarrow SVD(\mathbf{Z}_\mathcal{N}^T Y_\mathcal{N})$
$\mathbf{B} \leftarrow \text{eye}(d \times d)$
$diag(\mathbf{B}) \leftarrow \left[\frac{\text{range}(\mathbf{Y}_\mathcal{N}^1)}{\text{range}(\mathbf{Z}_\mathcal{N}^1)} ... \frac{\text{range}(\mathbf{Y}_\mathcal{N}^d)}{\text{range}(\mathbf{Z}_\mathcal{N}^d)} \right]$
$\mathbf{T} \leftarrow \mathbf{U}\mathbf{V}^\mathbf{T}$
$y \leftarrow x\mathbf{V}_{1...d}$
$y \leftarrow \mathbf{B}y\mathbf{T}$

return y

4.5 Sampling

4.5.1 Methods design

Since both the sampling methods proposed here have been conceived by us specifically to solve the problem of sampling from an unknown manifold, and since they are not standard sampling methods, a more extensive design section is required.
The first method proposed is *Nearest-Neighbours Interpolation* (NNI): this method relies on the assumption of local linearity of the manifold. To sample a new point, we randomly select a point in the dataset, we find its k nearest neighbours and we assume the manifold to be linear in the region of space around the neighbourhood; with this assumption in mind, the sampling of a new point is performed as described in algorithm 4. In this method there are two design choices that are worth exploring:

- The first one is the choice of the algorithm that randomly selects the new middle point between any two points. This can be a number from a uniform distribution from 0 to 1, or a truncated Gaussian distribution. While the first one ensures that every new point will be perfectly randomly placed in the outline defined by the k neighbours, it can also pick points that are "too close" to existing points. The truncated Gaussian would solve such problem, but picking repeatedly from a given neighbourhood would concentrate too many new points around its centre. Further testing is required to establish whether the problem of the first method is relevant in practical applications, or if the uniform distribution is sufficient.

– The second important parameter is the number of neighbours k: this is probably the most crucial parameter of this algorithm, the choice of k influences the area that will be available for sampling. A low value of k might induce the algorithm to ignore some areas of the manifold, while a too high value of k will cause it to pick points in regions of the manifold where there was not any point originally and there is also a higher chance of violating local linearity. Both these cases are undesirable, and finding the correct values of k will require further investigation.
In Figure 2, we can see that with $k=2$, some areas of the neighbourhood would be ignored and no points would be sampled there (e.g. the triangle CDF), with $k=3$ the neighbourhood is fully covered, but if k is too high, such as $k=6$, then the sample area is too large and includes region of the space that were originally empty (e.g. the triangle $D_2C_2G_2$).
At this stage a variant of the algorithm can also be taken into consideration: instead of building the neighbourhoods and sampling from them, an alternative idea might be to create the neighbourhoods first and then sample from smaller regions, like triangles, within the neighbourhoods. This ensures that the whole manifold is covered, but also that no unwanted areas are picked (given a low enough value of k). The construction of the triangles can be done with Delaunay triangulation [28].

The second method is *Distribution Fitting*: this algorithm aims to find a known distribution capable of accurately describing the data. If such distribution can be found within a given accuracy, the sampling becomes trivial as it is just a matter of sampling from a given distribution. There are only three relatively small design choices here: (i) the list of distributions, the list of all distributions on Scikit-Learn will be used for this task; (ii) the required accuracy of a distribution to be considered "good enough" to describe the data; and (iii) this method might suffer in the case where the data are perfectly describable by a distribution, but only in clusters (e.g. if the data are a set of Gaussian blobs, a Gaussian distribution can describe each blob but not all of them at the same time). For this reason the data will be analysed with the methods described above and, if necessary, it will be clustered before the application of this algorithm [21].

Last but not least, let us also recall that, in the case a Variational Autoencoder should be chosen to reduce the dimensionality of the data, sampling becomes a trivial task, as the neural network will produce a mean and standard deviation vectors from which we can easily sample new points without any additional work [15].

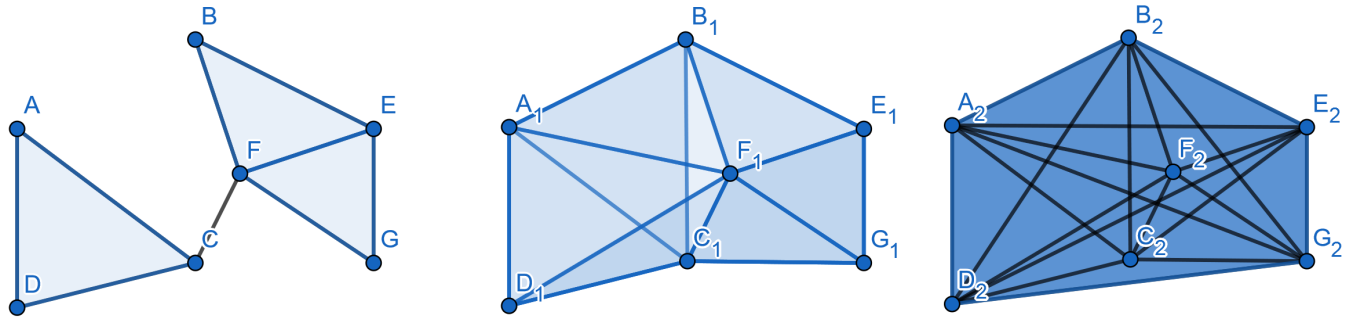


Figure 2: From left to right, a simple example of NNI with $k=2$, $k=3$ and $k=6$

Algorithm 4 Nearest Neighbours Sampling

Require: $\mathbf{X} \in \mathbb{R}^D, k$

$p \leftarrow$ pick_random($\mathbf{X}$)

$neighbours \leftarrow$ knn(p,$\mathbf{X}$,k)
for n in neighbours **do**
    $p \leftarrow p+(n-p)*$random(0,1)
end for

return p

4.5.2 Testing design

The next step is to find a way to evaluate the performance and correct working of both algorithms. To approach this task we will use two different methods: each method has its own advantages and disadvantages, but used in unison they should be able to correctly evaluate the proper working of these algorithms.
The methods are the following:

1. Visual test: a first qualitative approach is to cherry pick some selected or appositely generated datasets with specific characteristics that might cause the algorithms to fail (See Figure 3 for more details), and then visually check whether the algorithms failed in practice or not, or for which parameters they are more likely to fail. This test, for obvious reasons, can only be performed on 2D or 3D sets, and does not produce a numerical value to asses the methods, but, if the datasets are picked correctly, it can still give a helpful insight on the flaws and advantages of the methods used.

2. Error measure: a quantitative analysis consisting of using both methods to sample n new points (where n is the size of the dataset of origin) and then obtaining a numerical value describing the accuracy of the sampling using Bins Absolute Error (a measure of error defined appositely for this purpose, see the section below).

For these reasons, the use of both methods in tandem on some chosen datasets will be used to evaluate the performance of the sampling algorithms.

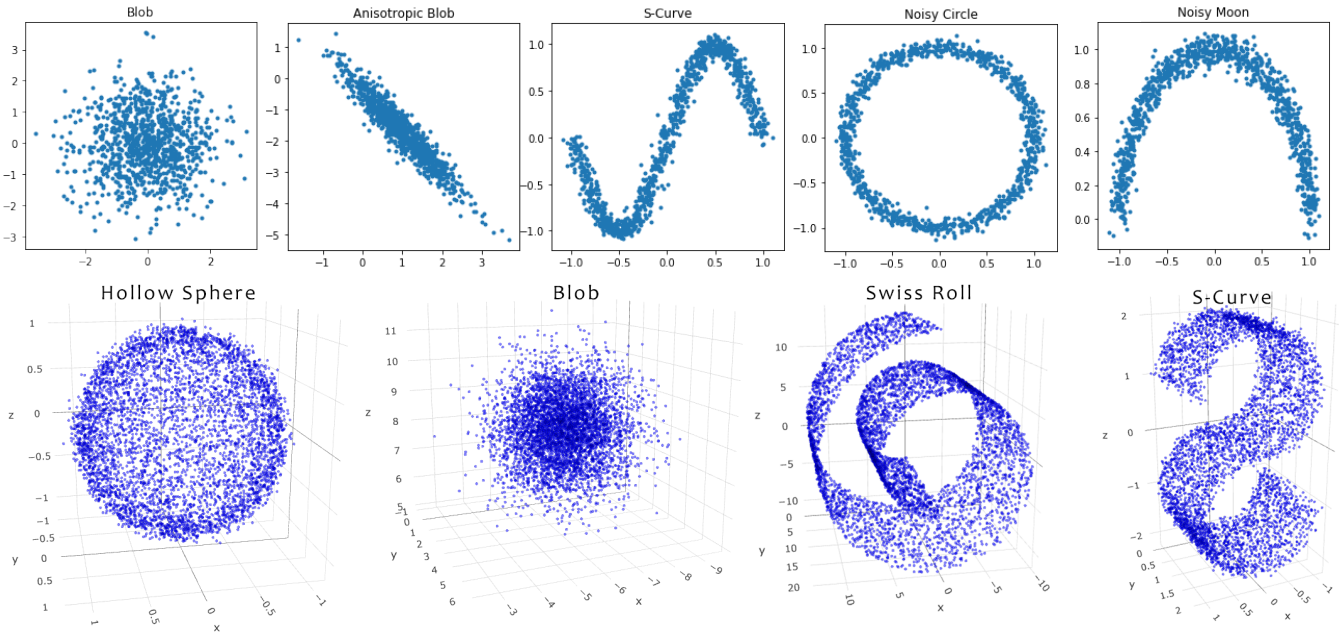


Figure 3: Some datasets picked and generated with the specific purpose to make the sampling algorithms fail because of their shape.

4.5.3 Bins Absolute Error

Since the sampling task as defined here is not a common task, finding a numerical measure of the accuracy of the sampling is a harder task than one might expect. Measures like MSE do not work when the points to compare are not in a specific order, or if it is not possible to pair them a priori in any way (Figure 4 shows a simple example of a case where MSE might fail). Therefore, a new algorithm has been developed.
The algorithm proposed to solve the problem has been called Bins Absolute Error because it is based on dividing the space of the original dataset in a grid (or in bins) and then assigning each point in both the original dataset and the sampled dataset to a bin, by comparing the number of points in each bin we obtain a numerical measure of error of the distribution between the two datasets.
It is important to notice the choice of error used: using an absolute measure of error allows us to know the theoretical maximum value of the error, using which it is then possible to express the error as a percentage.
In short, it can be summarised as follows: having an original dataset $\mathbf{X}$ and a sampled dataset $\mathbf{Y}$ such that $|\mathbf{X}|=|\mathbf{Y}|$

1. *Construct a grid around* $\boldsymbol{X}$ *so that in each square/cube/hypercube there are* $\boldsymbol{n}$ *points on average.*

2. *Assign each point in* $\boldsymbol{X}$ *to a bin.*
3. *Assign each point in* $\boldsymbol{Y}$ *to a bin.*
4. *Compute the absolute difference of the bins of* $\boldsymbol{X}$ *and the bins of* $\boldsymbol{Y}$*.*
5. *The result is given as* $|bins_x - bins_y|/(2N)$ *where* N *is the number of points in* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$*.*

The main advantages of this method are that (i) it is a *local measure*: it is a sum of local differences, rather than a global measure; (ii) it takes density into account: by computing the difference in the number of points in each bin we take into account, not only the positions of the points, but also the density of points in each area of the grid; (iii) and lastly, since a theoretical maximum of the error is known (i.e. in the case where there is no intersection in the assignment to bins between the points in $\mathbf{X}$ and in $\mathbf{Y}$ the error has value 1), we can express it as a percentage, where 0% represents a perfect sampling.
The method has also a disadvantage: when applied to dataset with really high dimensionality, in order to fulfil the requirement of having n points in each "square" of the grid on average, the total number of bins is calculated as $b = int(\sqrt[D]{N/n})$, where D is the number of dimensions of the dataset and $N = |\mathbf{X}|$. It is easy to see that if D is large b tends to 1. To solve this problem two solutions are proposed: (i) set the number of divisions on each axis to a fixed number regardless of the average number of points; or (ii) apply the division to only one dimension at a time, taking "slices" (in D dimensions) rather than bins, apply BAE on the slices, and repeat for all the dimensions. The average of all the individual results divided by the number of dimensions gives a better metric when D is high. This measure will still invariably increase as the number of bins or slices increases. To mitigate this effect we might divide the result by the number of divisions on each axis, which would make the value no longer between 0 and 1, but it should give a more accurate and scalable measure, this second version will be called BSE (Bins Scaled Error).

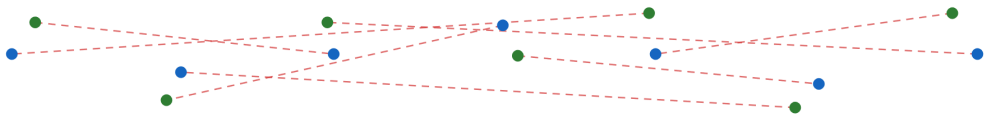


Figure 4: In blue the original points, in green the new sampled points. If the points are paired incorrectly in the two datasets MSE can fail by returning a much higher value than expected.

4.6 Pre-images problem

Since most of the literature behind the pre-images problem has been reviewed in Section 2.4, we will just propose a method to evaluate the performance of the mentioned pre-images problem solution (Autoencoders [14]). The method is quite straightforward and is based on calculating an error between the original dataset and a dataset created by reducing and increasing the dimensionality of the data.
Notice that it is also possible to train a neural network to solve the pre-images problem on the low and high-dimensional representation of the same points. Using this approach the low-dimensional points can be produced with any other dimensionality reduction method [20].
In short, the evaluation will proceed as follows:

1. *Consider a dataset* $\mathbf{X}$
2. *Using one of the methods above (using an Autoencoder if available or training a network if not) we produce* $\mathbf{Y}$*, the low-dimensional counterpart of* $\mathbf{X}$
3. *Using the proposed method we recreate* $\mathbf{X}'$ *from* $\mathbf{Y}$
4. *We can now evaluate the performance of the method as a an error measure between* $\mathbf{X}$ *and* $\mathbf{X}'$*. This can be done easily with Mean Squared Error or Bins Scaled Error.*

Using this method we can numerically estimate the accuracy of both procedures and also use it to tune the parameters of the decoding part of an Autoencoder.

4.7 Results evaluation and fitness for purpose

Now that each method has been independently evaluated, it is also necessary to design a procedure to evaluate whether the use of all the methods described above produce better results than the methods already in place to solve the same problems.

Since the aim of the project is to create points as similar as possible to hypothetical points generated by the original ESN, one way to evaluate the accuracy of the process as a whole is to compare the points generated using the methods proposed here and the points produced using the methods proposed by Ceni et al. [1] with new points produced by the original ESN to obtain a measure of the inaccuracy of the data produced with both methods.

5 Development

5.1 Clustering

The implementation of both the DBI index and the Elbow method followed to the letter the pseudo-code provided in Section 4.2. The development of the test consists mainly in plotting the result of both method on the same plot, which involved rescaling the results of the two method along the y axis to make them plottable using the same scale. This does not affect the validity of the results as we are only interested in minima and discontinuities, which are not affected by a linear rescaling along the y axis.
Once the rescaling has been performed, the result of the application of the methods is simply visible from the produced plots.
The method itself does not return a value for the correct number of clusters for several reasons: the data might not be cluster-able, the DBI and Elbow method might disagree, but if the data are clearly divisible in clusters the answer is obvious from the plot (see the Testing section for more examples).

5.2 Dimensionality reduction

As Scikit-Learn already provides a large library of dimensionality reduction tools, no implementation was required for PCA, KPCA, Isomap and LLE, although, it was necessary to decide the correct combination of hyper-parameters for KPCA (the kernel type), Isomap and LLE (number of neighbours used in the construction of the graph). This was done through cross validation. Moreover, Scikit-Learn also provides a method to calculate the residual variance after the application of PCA, allowing us to find the correct number of dimensions for each dataset with ease.

As a first step of the implementation of the tests described in Section 4.3.3, it was needed to determine a "base level" classification accuracy of the classifier for each dataset in order to compare it with the accuracy after the reduction. This task was performed using a standard implementation of 3-Nearest-Neighbours Classifier, and using 20% of each dataset as a test set to calculate the accuracy of the classification. The process was repeated 100 times with random splits of the datasets in order to reduce as much as possible any error due to an incorrect initialisation of the classifier.

Secondly, it has been necessary to devise a flexible implementation of both a standard Autoencoder and a Variational Autoencoder. Such implementation was developed keeping in mind the basic interface of the other dimensionality reduction methods (i.e. methods like `fit()` or `fit_transform()`). The implementation therefore required the construction of an `Autoencoder` class with the mentioned methods as well as parameters to decide the type of Autoencoder, the number of hidden layers, the number of epochs and the batch size of the training process. The class also exposes methods to embed new points in a manifold and to decode low-dimensional points after the chosen Autoencoder has been trained. In the case of a Variational Autoencoder, a method to sample new points has also been exposed.
For the specific implementation of the neural network, Sequential layers were used, as well as the Adam optimiser [29], MSE as loss metric and Relu as activation functions. The reason of these choices is because an Autoencoder can be thought as a regressor network and therefore the guidelines of Ketkar et al. were followed [30].
The sampling of new points using a Variational Autoencoder was performed as follows:

1. *Pick a random entry from the means and standard deviations vectors produced in the training process.*
2. *Sample from a Gaussian distribution with the chosen parameters.*
3. *Repeat for all the new points to sample.*

Finally, the last main step before implementing the tests involved writing the code so that we can call every dimensionality reduction method with the same procedure. To do so a `parameters` dictionary was built to store the parameters for each dataset and each method, allowing us to initialise every method as `method(**parameters[method][dataset])`. Once this has been done, the development of the test consisted in reducing the dimensionality, for each method of each dataset (by using the exposed method `.fit_transform(dataset)`), then apply the 3-Nearest-Neighbours Classifier to the resulting low-dimensional representation and calculate the difference

in accuracy between before and after the reduction.
A method to cross validate parameters has also been developed to allow for faster testing of different hyper-parameters. The method changes the `parameters` object for all the values in a given list and performs the reduction with all the different parameters, logging the results. The changed parameter can be relative to the single dataset analysed (k in Isomap, for instance, is different for each dataset) or to the method in general (the type of Autoencoder is a parameter valid for all the datasets).

5.3 Out-of-sample extension

First of all, the Generalised Out-of-Sample Extension (GOoSE) method was implemented following the guidelines outlined in Sec. 4.4 with no major differences or obstacles.
It was then necessary to implement a neural network capable of learning the dimensionality reduction performed by another method in order to apply it to new unseen points. This was achieved by constructing a NN with a structure similar to the encoder part of an Autoencoder and then training it on a previously constructed low-dimensional dataset with the same optimiser and loss metric of a standard Autoencoder.
To test the accuracy of the two methods, the following procedure has been adopted: using all the dimensionality reduction methods, the original dataset was first reduced to a lower dimensionality, then both methods (GOoSE and neural network) were trained on a portion of the original and low-dimensional set (80%), and finally the remaining 20% was embedded using the two methods. By using MSE, it was then possible to compare the accuracy of the embedding.
Finally, for some specific datasets, like the Blobs dataset, it is also possible to plot the result of the dimensionality reduction and the successive embedding of the test set.

5.4 Sampling

As per Section 4.5.1, two main methods needed to developed: Nearest Neighbours Interpolation and Distribution Fitting.
The implementation of the former quickly branched into two different algorithms: the first one following the main implementation outlined in Section 4.5.1, and the second one involving the construction of a triangulation through Delaunay Triangulation.

5.4.1 Standard NNI

The implementation of the standard NNI followed the pseudo-code provided with just a small difference: once a neighbourhood has been built, the points in the neighbourhood are sorted by distance from the picked point (closest to furthest). A parameter has also been added to add noise to the produced sampled data.
At the development stage some short experiments have been performed to choose what function to use to get a random point between any two given points. After the tests, both visual and using BAE, it was decided to use the truncated normal with a relatively high standard deviation; this was done to obtain points statistically closer to the mean of the two original points, but not close enough to cause a problem with many subsequent sampling from the same neighbourhood. Moreover, this choice produces "better distributed" points, allowing the algorithm to fill the gaps between the original points in the neighbourhood better than with a uniform distribution.

5.4.2 Delaunay Sampling

Another option similar to NNI is to divide the space in N-dimensional tetrahedrons using the Delaunay Triangulation algorithm [28].
Because of how the algorithm constructs the triangulation, it also creates triangles where no points are present. The following procedure allows us to take only the triangles in populated regions of the space:

1. *Construction the triangulation (Fig. 5, left).*
2. *Sort the tetrahedrons by their size (a measure based on how spread-out the vertices of the tetrahedron are).*
3. *Based on the idea that "illegal" tetrahedrons connect points far away from each other, we can sort them based on their size. Plotting the sizes we can see that there is a point after which the sizes sharply increase.*
4. *We can either set a percentile-based threshold, taking the smallest x% tetrahedrons, or a gradient-based threshold. The latter can be done by calculating the local gradient in every point of the plotted curve and therefore find the discontinuity, excluding all the tetrahedrons after it (Fig. 5, right).*
5. *To sample new points we can pick a tetrahedron with a probability inversely proportional to its size (closer points produce smaller sizes by definition) and then apply NNI on the points of the tetrahedron as if it was a neighbourhood.*

This method allows us to exclude the wrong tetrahedrons and to sample new points with a density based on the original distribution. A downside of the method is that it has a high number of hyper-parameters compared to NNI (cut-off threshold and scaling factor for the size in the probabilistic choice, in addition to the hyper-parameters of NNI itself). For later reference, the gradient threshold has been set to 0.002 and the probability distribution function has been set to $f = 1/\text{sigmoid}(size)$ in order to pick the larger triangles a little more often.

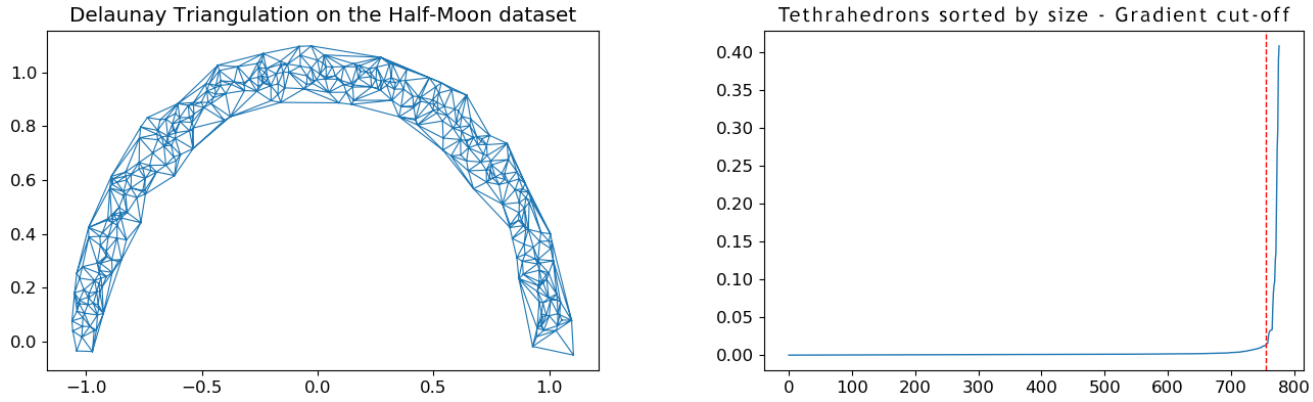


Figure 5: Visualisation of the Delaunay Triangulation in 2D (left) and the gradient-based cut-off method (right).

5.4.3 Distribution Fitting

Several intermediate helper functions were needed to implement Distribution Fitting as described in the Design section.
First of all, for each axis, the data axis is divided in bins, and compared to a list of distribution probability functions for the same bins. By doing so, the best distribution and the best parameters are found, and a function to describe each axis is chosen. The error measure used is the sum of squared errors between the bins of the original data and each fitted distribution.
Finally, following the chosen distributions and parameters, new points are sampled with an appositely developed function.

5.4.4 Datasets creation

The last step before being able to write the tests is to develop functions to obtain the datasets in Fig. 3.
The implementation of "Blobs", "Noisy Circle", "Noisy Moon", "Swiss Roll" and the 3 dimensional "S-Curve" was facilitated by the tools provided by Scikit-Learn, which produced similar datasets that just needed to be elaborated. The Noisy Circle dataset, for instance, was originally composed of two concentric circles that needed to be separated to obtain a single circle.
The creation of the other datasets needed some more work: the 2D S-Curve is simply a truncated sinusoidal function with added parametric noise; the Anisotropic Blob is a standard blob multiplied by a transformation matrix that stretches the shape of the blob; Hollow Sphere needs a more in depth explanation which can be found in Algorithm 5.

Algorithm 5 Make Hollow Sphere implementation

Require: *number_samples*,*radius*,*noise*

$\phi \leftarrow \text{uniform\_random}(0, 2\pi, number\_samples)$
$\theta \leftarrow \text{truncated\_normal}(0, \pi, number\_samples, mean = \pi/2, std = .25\pi)$

$x \leftarrow r\cos(\phi)\sin(\theta)$
$y \leftarrow r\sin(\phi)\sin(\theta)$
$z \leftarrow r\cos(\theta)$

$sphere \leftarrow (x, y, z) + \text{Gaussian\_noise}(noise)$

return *sphere*

5.4.5 Bins Absolute Error and Bins Scaled Error

In the development of both BAE and BSE, the first step is to devise a way to divide the space so that every region contains n points on average. To do so the minimum and maximum on each axis are found and an hyper-cube is constructed around the points. Each axis is then divided in $\sqrt[D]{N/n}$ parts, where D is the number of dimensions of $\mathbf{X}$ and N is the number of points in $\mathbf{X}$.
In the case of BSE, every axis, one at a time, is divided in a fixed number of parts and the difference of assignments is calculated for each dimensions and then averaged. This measure also takes into account the number of partitions and the number of dimensions of $\mathbf{X}$.
A note on the Python implementation: the counting is performed through dictionaries because of their performance advantage over other methods (accessing an element has complexity of $\sim O(1)$).

5.4.6 Testing suite

As per Section 4.5.1, it is first necessary to cross validate the parameter k in NNI to find the best value of the parameter. This is done by plotting the average BAE of NNI on all the datasets while varying the value of k and picking the k corresponding to the lowest error. Now that all the basic elements have been developed, it is possible to write the class to perform the tests and to analyse the performances of each method. First of all, a function to visualise the grid division of BAE has been written to make the algorithm more intuitive for the reader. Secondly, wrapper functions with the support for 2D and 3D plotting for each method have been developed. Finally, a function to apply the method to all the datasets, both in 2D and 3D, has been implemented to allow us to test all the methods with a single function call. Plotting of the sampled points is also supported in such function.
A wrapper function of the sampling using a Variational Autoencoder is also present in the `Sampling` class, concentrating all the methods related to sampling in the same structure.

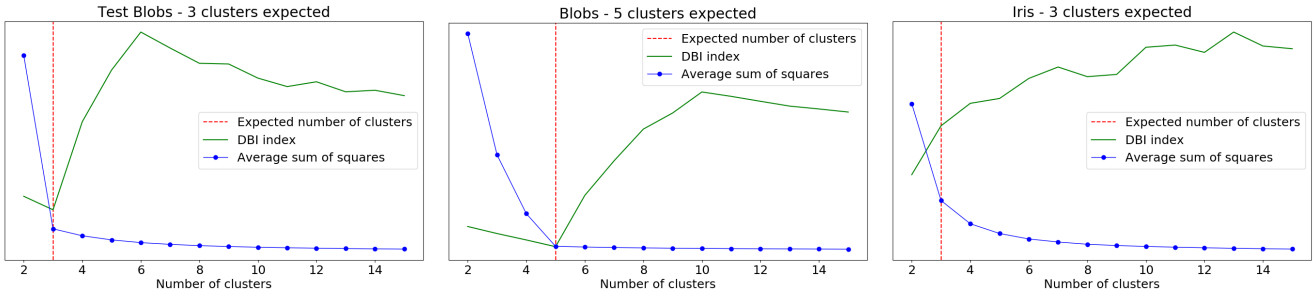
5.5 Pre-images

The implementation of the neural network used to solve the pre-images problem is a mirrored version of the one described to solve the OoSE problem.
The testing function trains the neural network on a set made of the low and high-dimensional representation of the same data (where the low-dimensional representation is produced with one of the method in Section 4.3). Then, a test low-dimensional set is passed to the network and the new high-dimensional points are compared with the original dataset using MSE and BSE. The process is repeated for all the datasets in Table 2 with low-dimensional points produced with each dimensionality reduction method.

6 Testing

6.1 Clustering

The testing for clustering consisted mostly in checking the correct implementation for both the Davies-Bouldin index and the Elbow method. As explained in the Design section, the main test performed had the aim to ensure the correct functioning of these two methods by checking that we get the expected results when applied to a set of three isotropic blobs in three dimensions.
In addition, we have also applied the same two methods to some of the datasets proposed in Table 2 getting results that match the specification of the dataset. A plot of the results can be found in Fig. 6a for the base test case with blobs. A more complex case with five blobs in 200 dimensions can be found in Fig. 6b, and finally Fig. 6c shows the application of both methods on the Iris dataset.



(a) Results on three isotropic blobs **(b)** Results on the Blobs dataset **(c)** Results on the Iris dataset

Figure 6: Result of the application of the DBI index and Elbow method on three datasets

The tests produced an output consistent with the expected number of clusters. Both method appear to be working without any problem in the case of distinct Gaussian blobs. It is to be noted that the DBI index fails to recognise the right number of clusters in the Iris dataset, whereas, for the same dataset, there is a discontinuity in the plot of the Elbow method (showing that the right number of clusters can be found with the Elbow method in this case). Nonetheless, as expected, the conjunct use of both method did produce the hoped result.
One flaw of both methods, not visible from these plots, is their inability to discern non Gaussian clusters. This is due the fact that they both rely on a measure of compactness of the clusters (rather than connectivity), which is a characteristic of Gaussian blobs. Some other clusters, like two concentric circle, will not be detected using these methods.

6.2 Dimensionality reduction

6.2.1 Finding the number of dimensions

The testing for all the dimensionality reduction methods was conducted in two different parts. The first part had the aim to determine the correct dimensionality to use in the reduction for each dataset, which has been done using residual variance on a reduction performed with PCA. We have applied this method to all datasets in the list proposed in the Design section, obtaining a "correct" number of dimensions to reduce each dataset to, while preserving a target relative variance. An example of the result of this process is can be found in Fig. 7.

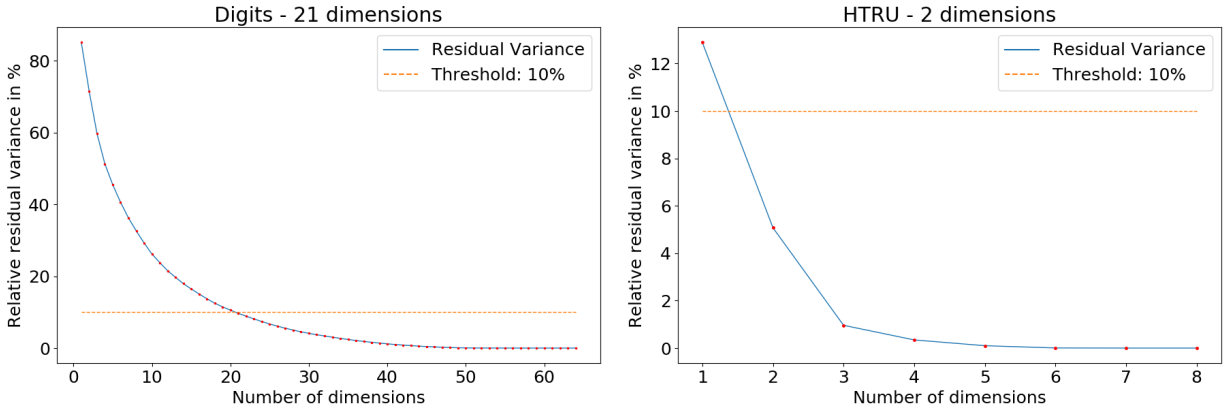


Figure 7: Plot showing the use of residual variance in conjunction with PCA to find the correct number of dimensions for the reduction while preserving 90% of the variance

6.2.2 Reducing the dimensionality of datasets

The second part consisted in testing the dimensionality reduction methods with the procedures discussed in Section 4.3.3. This second part aimed to find a numerical value to describe the loss in information content after the reduction process using the difference in accuracy of a simple 3NN classifier. The results in Table 3 show, for each dimensionality reduction method, the number of initial dimensions, the number of target dimensions (with $d \ll D$), the accuracy of the classifier before the application of the dimensionality reduction and the difference of accuracy after the reduction. The difference is given as an average over several trials with the respective standards deviation and lowest value reached throughout all the trials.
There are a few notes to be made:

- Some minimum values are negative: this means the classifier performed better after the reduction than it did before. This is possibly partly due to the oscillations due to the stochasticity of the classification algorithm used, but it could also mean that the reduction process helped the classifier in correctly labelling the data.
- Isomap appeared unable to reduce the dimensionality of the HTRU dataset as it always ran out of memory on all machines used for the tests.
- For the methods that have hyper-parameters, the result of the best combination of them is reported in the table below.
- In Table 3 the best (green) and worst (red) method for each dataset have also been highlighted.

Dataset	D	d	Base accuracy	PCA			KPCA			ISOMAP		
				avg	std	min	avg	std	min	avg	std	min
Digits	64	8	98.75%	2.49%	1.01%	0.01%	2.76%	0.89%	0.01%	-0.19%	0.35%	-0.97%
Faces	4096	16	85.05%	4.08%	4.68%	-0.09%	4.99%	3.63%	0.00%	12.05%	3.88%	3.80%
Blobs	200	2	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	8.75%	1.54%	7.00%
Parkinson	753	1	72.08%	0.59%	3.50%	-0.06%	0.86%	3.53%	-0.08%	0.30%	3.04%	-5.55%
Madelon	500	3	69.84%	2.53%	2.06%	-0.02%	2.48%	1.28%	0.00%	4.17%	1.60%	1.19%
HTRU	8	2	97.15%	0.98%	0.25%	0.00%	1.38%	0.26%	0.01%	NA	NA	NA

Dataset	Base accuracy	LLE			Autoencoder		
		avg	std	min	avg	std	min
Digits	98.75%	1.85%	1.07%	-0.14%	1.32%	0.21%	0.98%
Faces	85.05%	0.74%	5.31%	-3.70%	0.64%	3.53%	0.48%
Blobs	100.00%	3.04%	5.10%	0.00%	0.00%	0.00%	0.00%
Parkinson	72.08%	-0.52%	2.72%	-7.53%	-1.18%	2.33%	-6.50%
Madelon	69.84%	2.43%	1.57%	-2.27%	11.51%	1.64%	9.60%
HTRU	97.15%	0.22%	0.26%	-0.23%	0.28%	0.37%	-0.22%

Table 3: Results of the application of different dimensionality reduction methods to a variety of datasets.

From Table 3 we notice that all the methods performed well in general, without any major discrepancy between the result of the various methods. Nonetheless, some exception can be noticed:

- While being generally good, the Autoencoder failed to find a good low-dimensional representation of the Madelon dataset in all the runs of the algorithm.
- Both LLE and Isomap failed in reducing the Blobs dataset. This is to be expected and does not contradict what can be found in the literature about these two methods. The reason of failure resides in their dependence on the construction of a neighbourhood graph, which leads to major errors when the data are condensed in a small number of regions in space (i.e. clusters).
  It is to be noted that both methods did produce a perfect low-dimensional representation (according to the evaluation method used) when presented with each cluster separately.
- Isomap, while also performing significantly worse than all other methods, failed to reduce the HTRU dataset due to apparently excessively high memory requirements on all machines it has been tested on. This can be attributed to the construction of a large k-neighbourhood graph and the calculation of a large number of geodesic distances.
  This latter fact alone makes it unfit for the original purpose as the number of points to reduce is not known a priori and robustness is key in the project.
- From this first analysis it would appear that LLE is the most robust and reliable method, always capable of producing a good low-dimensional representation regardless of the dataset. Although, the Autoencoder is the one that produced the best reduction more often.
- Variational Autoencoders are not present in the table as their results are not substantially different from standard Autoencoders (displaying the same weaknesses on the Madelon dataset), and therefore have been omitted for brevity.

6.2.3 Autoencoders

A separate note is required for the tests of Autoencoders. As they are not substantially different from what has been described in Section 4, we will just proceed to provide the results of the experiment.
In particular, we are going to analyse the effect of changing each hyper-parameter individually on the performance of the Autoencoder and then plot the result of the test. The aim is to find a correlation between the change of some hyper-parameter and performances.

Batch size	Digits		Faces		Blobs		Parkinson	
	avg	std	avg	std	avg	std	avg	std
32	1.69%	0.73%	6.03%	0.59%	0.00%	0.00%	0.34%	3.03%
64	2.25%	0.35%	6.18%	0.94%	0.00%	0.00%	2.37%	2.11%
128	0.95%	0.60%	6.50%	0.78%	0.00%	0.00%	1.26%	5.37%
256	0.95%	0.60%	6.35%	0.84%	0.00%	0.00%	-1.37%	2.19%
512	1.97%	0.92%	5.93%	0.89%	0.00%	0.00%	-0.45%	4.66%

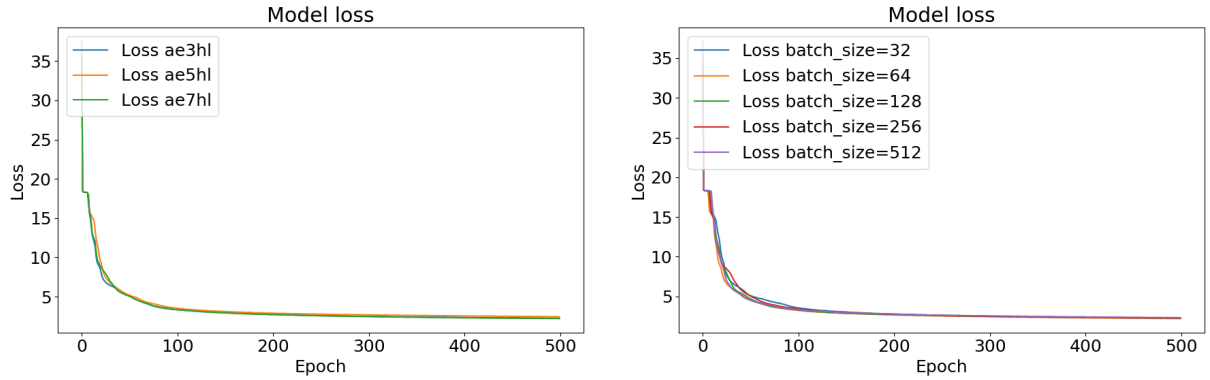
Batch size	Madelon		HTRU		Averages	
	avg	std	avg	std	avg	std
32	9.50%	1.71%	2.51%	2.48%	3.34%	1.42%
64	10.92%	3.08%	2.15%	2.28%	3.98%	1.46%
128	10.42%	1.69%	2.09%	2.16%	3.54%	1.77%
256	11.27%	1.56%	3.76%	2.57%	3.49%	1.29%
512	11.73%	1.70%	1.61%	2.22%	3.46%	1.73%

Table 4: Differences in accuracy varying the batch size during the training of the Autoencoder

Hidden Layers	Digits		Faces		Blobs		Parkinson	
	avg	std	avg	std	avg	std	avg	std
3	2.23%	1.34%	6.05%	1.02%	0.00%	0.00%	-2.03%	3.50%
5	3.34%	1.52%	6.40%	0.71%	0.00%	0.00%	0.87%	1.69%
7	1.95%	1.24%	6.23%	0.82%	0.00%	0.00%	-1.89%	2.65%

Hidden Layers	Madelon		HTRU		Averages	
	avg	std	avg	std	avg	std
3	11.88%	4.15%	0.66%	0.51%	3.13%	1.75%
5	11.88%	1.28%	2.61%	2.36%	4.18%	1.26%
7	13.19%	3.45%	1.24%	2.47%	3.45%	1.77%

Table 5: Differences in accuracy varying the number of hidden layers in the Autoencoder



(a) Loss of autoencoders with different numbers of hidden layers

(b) Loss of autoencoders using different batch sizes in the training process

Figure 8: Comparison of model loss of Autoencoders with variable parameters

From an analysis of Table 5, which shows the effect on changing the number of hidden layers on the performances, it is possible to notice that the number of hidden layer does not affect the performance of the network in the test setting proposed. This is due to the fact that even with three hidden layers the network appears to be capable of modelling even complex manifolds just as well as a network with five or seven layers.
From Table 4, it is also possible to see how the batch size also does not affect the performance of the Autoencoder

as much as it was originally expected.
Moreover, Fig. 8a and Fig. 8b confirm the above hypothesis by showing the two parameter have no impact on the loss in the training process of the network for this specific case.

6.3 Out-of-sample extension

When testing the two different proposed methods (GOoSE and Neural Network) used to solve the Out-of-Sample Extension problem, we tried reducing the dimensionality of a test set for which a low-dimensional representation had already been found. For the preliminary reduction all the methods above have been used, including Autoencoders. Even though this may seem redundant, we want to be able to check whether a neural network is able to model a function capable of reproducing not only the reduction performed by methods, such as LLE and Isomap, that are both non-linear and connectivity-based, but also the one produced by a different neural network.
The results shown in the table below compare the results of the use of GOoSE and the use of an appositely trained neural network (hence the name NNOoSE) in embedding new points in the manifold.

Dataset	**PCA**		**KPCA**		**Isomap**	
	GOoSE	NNOoSE	GOoSE	NNOoSE	GOoSE	NNOoSE
Digits	5.7E+14	4.86E-02	1.0E+14	3.39E-02	2.1E+15	1.97E+00
Faces	1.6E+05	1.41E+00	5.6E+03	7.23E-01	3.9E+05	9.98E+00
Blobs	5.9E+00	3.05E-02	9.0E-01	1.25E-02	6.4E-01	1.71E-01
Parkinson	7.2E+00	1.83E-01	1.6E-01	5.58E-02	1.8E+01	2.90E+00
Madelon	1.9E+00	1.06E-01	1.4E-01	8.25E-03	8.0E+00	4.64E+00
HTRU	8.7E-01	1.15E+00	1.5E+00	1.18E-01	NA	NA

Dataset	**LLE**		**Autoencoder**		**Averages**	
	GOoSE	NNOoSE	GOoSE	NNOoSE	GOoSE	NNOoSE
Digits	1.84E+04	1.24E-03	1.19E+15	3.21E-01	6.97E+14	5.13E-01
Faces	1.06E+03	7.12E-01	4.48E+06	1.02E+01	1.41E+05	3.21E+00
Blobs	1.12E-02	4.92E-05	1.42E+00	3.90E-02	1.87E+00	5.36E-02
Parkinson	1.87E-02	4.78E-03	6.46E+00	1.48E+00	6.42E+00	7.87E-01
Madelon	1.49E-02	7.68E-03	3.87E+00	1.68E+00	2.52E+00	1.19E+00
HTRU	3.83E-03	4.92E-03	7.07E-01	8.18E-03	8.06E-01	4.24E-01

Table 6: Mean squared errors of GOoSE and NNOoSE in embedding new points to lower dimensional representations created with different dimensionality reduction methods.

While GOoSE did produce acceptable results in the majority of cases, it appears to be failing in embedding new points in the Digits dataset, regardless of the method used to reduce the dimensionality (although LLE seems to be able to mitigate the problem to some extent). On the other hand, the neural network used in the process never failed to embed unseen points and seems to perform from marginally to significantly better than the GOoSE in all cases, sometimes by several orders of magnitude.

6.4 Sampling

6.4.1 Cross validation for parameters in NNI

As a first step, the parameter k for NNI has been chosen via cross validation using BAE as error measure. The optimal value according to the performed test is $4 \leq k \leq 6$. A plot of the result can be found in Fig. 9.

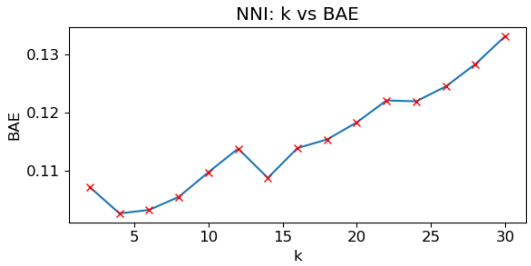


Figure 9: Cross validation for the parameter k in NNI

6.4.2 Performance comparison between NNI and DF

In order to test all sampling methods, we used the procedure described in Section 4: using NNI, Delaunay and DF, N points were sampled (where N is the number of points in a dataset) from some previously constructed distributions and the results were plotted. The corresponding Bins Absolute Error for each sampling has also been measured.

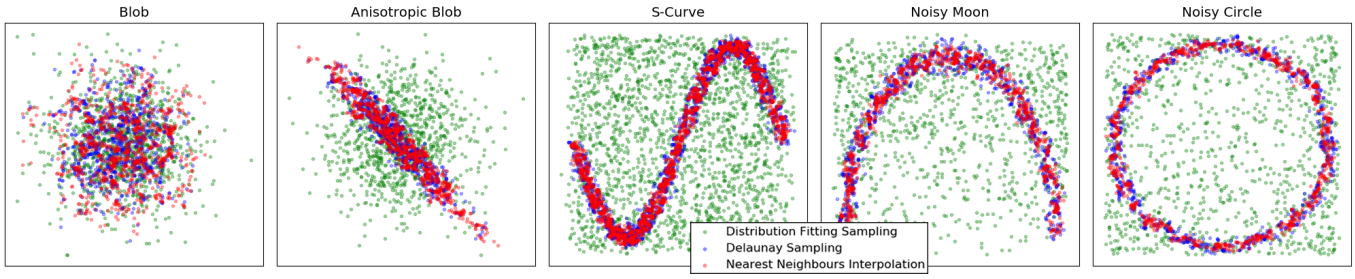


Figure 10: Results of the application of DF, Delaunay and NNI sampling to the chosen 2D datasets

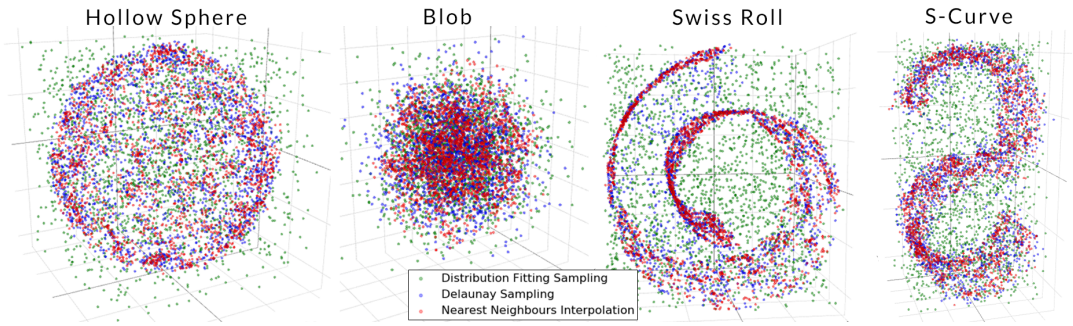


Figure 11: Results of the application of DF, Delaunay and NNI sampling to the chosen 2D datasets

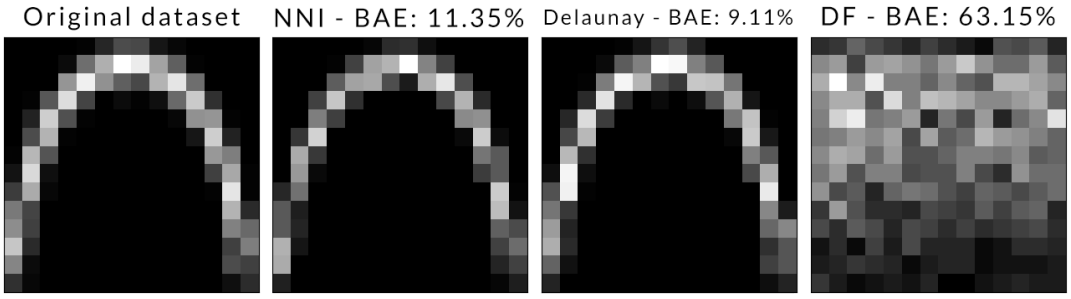


Figure 12: Example of the grid produced by Bins Absolute Error on the Noisy Moon dataset from left to right: original dataset, NNI, Delaunay and DF

Method	Half-Moon	Circle	Blob	Anis. Blob	S-Curve	Swiss Roll	S-Shape	Blob	Holl. Sphere
DF	65.62%	65.24%	10.64%	58.82%	73.44%	42.16%	42.28%	9.10%	50.36%
Delaunay	7.20%	8.02%	9.52%	5.48%	6.44%	10.04%	9.82%	6.32%	8.98%
NNI	8.86%	9.74%	8.90%	5.78%	8.84%	9.32%	10.38%	7.72%	9.14%

Table 7: Bins Absolute Error on the points from the chosen datasets

There is a number of things to mention from a first analysis of the above plots and table:

- Looking at Fig. 10 and 11, we have a first visual feedback on the performance of NNI (in red), DF (in green), and Delaunay (in blue). The original datasets can be found in Fig. 3. It is immediately clear from the plots that DF fails to sample all the non-trivial distributions, while NNI and Delaunay manage to perform within a reasonable error in every proposed dataset.
- Looking closely at Fig. 11, we can see that, despite the low value of BAE, Delaunay samples points not on the surface of the manifold when applied to the Swiss Roll dataset.
- Fig. 12 shows a false colour rendition of the grid produced during the calculation of BAE for the Noisy Moon dataset. In the images, a lighter colour represents a higher density of points in the square of the grid: black represents a density of 0 and white represents the maximum density present in the grid.
  Even though the central ones (NNI and Delaunay) more closely resembles the original one, it can be noticed that they do not perfectly mimic the original distribution (hence the non-zero error in Table 7).
- Table 7 shows the numerical value of the BAE calculated for the sampling of each dataset using all three methods. The result of the previous images is confirmed by the values reported here: NNI and Delaunay produce much lower errors than DF. Moreover, judging by the plots and by the table it seems that a small error might not necessarily be negative in the scope of evaluating the accuracy of a sampling method as the new points do not have to be exactly the same as the original ones.

Despite having mentioned it in Section 4, there is no reference to changing the function to pick an intermediate points between two points in NNI. The reason for this is that, after some tests in the development part of the project, only a small difference in favour of the truncated Gaussian could be found in the performance of the method, and therefore no further tests have been devised.

6.5 Pre-images

The table below shows the Mean Squared Error, for each dataset reduced with every dimensionality reduction method, between the original high-dimensional representation of a dataset and the one produced using an appositely trained neural network.

Dataset	**PCA**	**KPCA**	**Isomap**	**LLE**	**Autoencoder**	**Averages**
Digits	0.319	2.294	0.485	0.618	0.256	0.929
Faces	0.354	0.339	0.453	0.390	0.331	0.384
Blobs	0.075	0.363	0.560	1.000	0.051	0.500
Parkinson	0.716	0.840	0.924	1.118	0.731	0.900
Madelon	1.143	1.057	1.156	1.002	1.281	1.090
HTRU	0.952	0.221	NA	1.037	0.966	0.737

Table 8: Mean squared error obtained by increasing the dimensionality of low-dimensional representations of some chosen datasets using a neural network

From an analysis of Table 8 it would seem that the choice of the dimensionality reduction method used to produce the low-dimensional representation of the points affects the performance of the neural network. In particular PCA and Autoencoders help the NN to perform better while LLE is the one that the worst performances in general.

7 Evaluation

The evaluation of the overall procedure will proceed in two parts: a first part dedicated to applying all the above methods to the datasets of points generate by the ESN, with the aim of evaluating the performance of the combination of all the tools implemented and to choose the best combination of tools to use. A second part will involve comparing the points produced by reducing the dimensionality, sampling and inverting the reduction with the corresponding process performed using Ceni's tools (PCA, points sampled on a uniform hyper-grid, inversion using PCA) to verify that our procedure to create new points does indeed produce better results.

7.1 Applicability of Clustering

As outlined in Section 1.2, it is first necessary to verify whether the data are divisible in partitions and, if so, in how many partitions. This is done by using the DBI and Elbow methods, looking for a minimum in the DBI and a discontinuity in the Elbow plots.
The figure below shows the application of both methods to the points around one of the attractors.

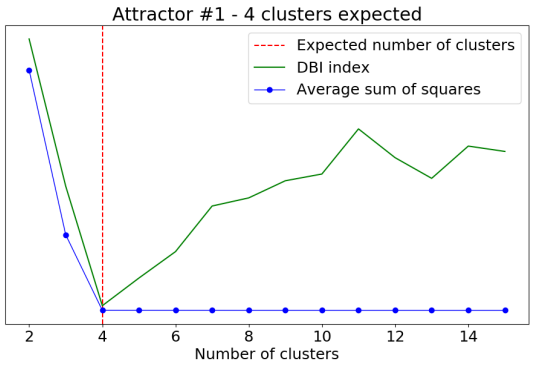


Figure 13: Application of Elbow method and DBI index to points around attractor 1

From the plot it is remarkably clear that the data are indeed divisible in clusters according to both the methods used. The plot also sets the number of clusters to 4, without any uncertainty.
Note that only one Attractor is reported here as all the others produces extremely similar results, and therefore the plots were omitted for brevity.
The points around every attractor will therefore be divided in 4 clusters using Spectral Clustering, as per Section 2.1.

7.2 Finding number of dimensions

Residual variance on a PCA reduction has been used to find the correct number of dimensions to reduce the dataset to. In particular, for the test we set the residual variance threshold to 5%.

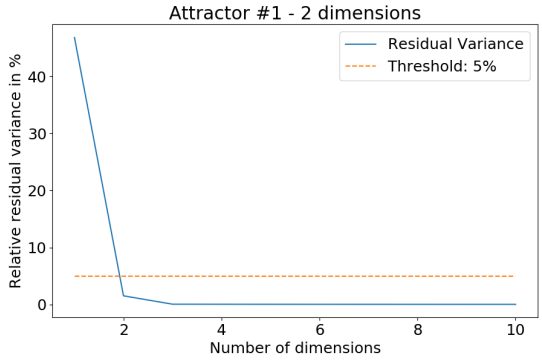


Figure 14: Residual variance test to find the correct number of dimensions while retaining 95% of the original variance according to PCA.

It is clear from the plot that the correct number of dimensions, while keeping 95% of the original variance, is equal to 2 and therefore 2 will be used in all the consecutive steps.

7.3 Dimensionality reduction

There are several factors to take into account when choosing the dimensionality reduction method. First of all, judging by Table 3, it is safe to say that we can reduce the choice to LLE and Autoencoder.
On one hand, Autoencoders have the possibility to perform better than LLE, like it did for some of the datasets, but it is also possible that they fail in reducing the dimensionality (like with the Madelon dataset), but Autoencoders also allow us to simplify the rest of the operations as they provide a native support for OoSE and pre-images. On the other hand, LLE seems to be more stable in reducing the dimensionality as it has never failed for any of the datasets, although, the high-dimensional representation produced from the points reduced LLE was the least accurate when using a neural network (Table 6).

Since deciding a priori might lead to some errors, it is better to perform all the operations with both methods and to compare the final results in order to decide which one to use.
The tests were performed using NNI for the sampling (the choice between NNI and Delaunay will be taken later, NNI is only used as a placeholder method for now). The sampling is then evaluated using BAE, the pre-images is performed with NNOoSE and the final high-dimensional points are compared both with the original ones and with a validation dataset made of points generated by the original ESN using BSE.
The error on the original dataset is calculated as the average of cluster-by-cluster errors (as the clusters assigned are the same for both sets), while the error relative to the validation dataset is calculated on the whole attractor, hence the two errors are only comparable with errors from the application of different methods. Comparing the error between the original dataset and the validation dataset does not provide any information.

Method	**BAE sampling**	**BSE original dataset**	**BSE validation dataset**
Autoencoder	1.39%	0.15522120	0.007412529
LLE	2.43%	0.17877023	0.009732382

Table 9: Comparison between Autoencoder and LLE on the points generated by the ESN.

From the table above it seems that LLE affects negatively the performance of the sampling method and of the pre-images process (respectively, making the sampling $\sim$75% worse and the pre-images $\sim$15% worse). For these reasons and the reasons above, Autoencoders will be used from now on to reduce the dimensionality of the dataset.

It is also worth comparing the performance of standard AEs with the one of VAEs: if the difference in performance is negligible, VAEs would allow us to sample new points without the need to use any additional sampling method.

Method	**BAE sampling**	**BSE original dataset**	**BSE validation dataset**
Autoencoder	1.39%	0.155221200	0.007469241
VAE	3.25%	0.186501671	0.012503259

Table 10: Comparison between Autoencoder and VAE on the points generated by the ESN.

Unfortunately the VAE performs much worse on this particular dataset, making it a worse option than a standard Autoencoder.

7.4 Out-of-sample extension

Since the difference in performances between the NN-based approach and the GOoSE approach has already been tested, there is no need to further test which method to use for the particular dataset at hand as the NN was generally better in every test. A neural network will be used to embed new points into the manifold.

7.5 Sampling

Analysing the figures and plots in Section 6.4, we can exclude Distribution Fitting, but the choice between NNI and Delaunay is not trivial.
From Figure 10 and 11, we can see that, visually, the two methods perform very similarly, and Table 7 confirms this observation. It is to be noted that Delaunay partially fails to sample new points on the surface of the manifold on the Swiss Roll dataset, probably due to a (possibly microscopical) error in the choice of the threshold parameter. Nonetheless, the BAE, when applied to the new sampling is still relatively low.
Another factor to take into consideration is which type of sampling allows the neural network to create better pre-images.
To test these possibilities, we use the same setting of the above sections, measuring the BAE on the newly sampled points and the BSE on the pre-images, both on the original dataset and on a validation dataset.

Method	BAE sampling	BSE original dataset	BSE validation dataset
NNI	1.98%	0.152740585	0.007657559
Delaunay	1.70%	0.164054765	0.011496254

Table 11: Comparison between NNI and Delaunay sampling on the points generated by the ESN and a validation dataset.

Despite Delaunay being better at sampling in lower dimensions, it seems that the sampling produced with this method impairs the application of the neural network to invert the dimensionality reduction process. Considering our aim is to obtain new points with a distribution as close as possible to the original points, we prefer to use NNI to solve the sampling problem.

7.6 Pre-images

As the only proposed method is the use of a neural network to solve the pre-images problem, and such method has been extensively tested in Section 6.5, there is no need to choose a best method to approach the problem in the particular case. Morevoer, the parameters have already been chosen through tests. For this reason further tests are not required in this section.

7.7 Comparison with existing method

Now that the best method to perform every task has been chosen, we can compare every step of Ceni's project, Section 4, with the same step but performed with the methods discussed above. To do so:

- We will reduce the dimensionality of the dataset of the PDVs with an Autoencoder, while Ceni used PCA.
- To sample new points we will need to apply NNI to the low-dimensional representation of the PDVs. Ceni used a hyper-grid to sample new points ([1], 4.2), and therefore we will compare the BAE of the result of NNI with the BAE of points sampled with the grid method.
- Then we will need to measure the accuracy of the out-of-sample extension process applied to both the new manifolds. We will use a neural network to do so, while Ceni used the matrix produced in PCA to solve the problem. To compare the accuracies we will use MSE between the expected low-dimensional points and the newly produced ones.
- We then proceed to invert the dimensionality reduction process on the sampled points by applying the chosen pre-images method. To invert the reduction Ceni used a method based on PCA. To evaluate this step we compare the BSE of the new high-dimensional points with both the original dataset and a validation dataset made of PDVs produced by the original ESN.

Method	BAE sampling	MSE OoSE	BSE original dataset	BSE validation dataset
Hereby proposed method	1.55%	0.012799516	0.154740210	0.008116015
Ceni's current method	95.69%	1.616045953	0.189360908	0.091490418

Table 12: Comparison between the methods proposed here and the currently implemented methods.

From the table above we can see that:

- The results of the BAE applied to the points sampled with the two proposed methods are notably different: while the method proposed here manages to sample new points with an error close to 0%, the grid method failed to produce a good sampling (according to the chosen metric). This is expectable as the grid method only takes into consideration the location of the points in space as a whole, and not the density of points in the region or the divisions in clusters. The worse results is explainable by recalling that BAE looks for discrepancies of density at a granular level and, even though the grid does cover the location of all points in the space, it also covers areas where none of the original points are present.
- The same reasoning can be applied to the OoSE problem: using PCA to embed unseen points will produce points in the close vicinity of the original points, and definitely on the same manifold, but possibly in areas of the manifold sparsely populated by other points. The poor results is again imputable to the low consideration given to density by PCA.
- The last two columns show that, while on the original dataset the difference in error of the pre-images process is relatively low, Ceni's method seems to fail in the generalisation process, producing points quite dissimilar in distribution to those in the validation set.

8 Conclusion

8.1 Aim of the project and intermediate steps

The paper set out with the aim of improving on Ceni's [1] results in the intermediate steps between the location of the attractors and the calculation of the excitability thresholds as described in his original paper. The aforementioned steps involved finding methods to massively produce points in the PDVs space extremely similar to those produced by the ESN in response to inputs during its training but without using the network itself.
In the context above, there was the need to reduce the dimensionality of the PDVs surrounding each attractor using a non-linear method that also allows an easy embedding of new points into the found manifold: this was accomplished by using an Autoencoder to reduce the dimensionality and a separate network to embed unseen points. The reduction was necessary to facilitate the application of the consequent methods.
Secondly, new points needed to be generated resembling as closely as possible the new representation found for the PDVs. This second task was performed by the means of a newly developed method: Nearest Neighbours Interpolation (NNI). To evaluate the performance of the sampling methodologies two new metrics that take local density into account were also developed: Bins Absolute Error and Bins Scaled Error. The proposed implementation of NNI is capable of producing a large amount of new points in a stochastic manner and was used to obtain a new set of low-dimensional points.
Lastly, to produce new points with the original dimensionality, a neural network capable of inverting the dimensionality reduction process was built and trained. When applied to the sampled points, the network was used to produce new high-dimensional points in the same region of space of the original PDVs dataset.

8.2 Summary of results

After the methods above were designed, developed and tested, they were finally applied to the PDVs dataset and each step was compared with Ceni's approach for the same step. While a direct comparison was not possible for all methods, it was possible, using BAE and BSE, to accurately compare the low-dimensional sampling and its high-dimensional counterpart.
The comparison in high dimensionality is performed on a validation dataset as well, yielding a result around 10 times better than the application of Ceni's methods on the same dataset. This last result confirms our original hypothesis: by bringing a significant improvement to every intermediate step, a considerable improvement in the final result follows.
As the new method of generating "synthetic PDVs" performs substantially better than the procedure already in place, it is reasonable to assume that every subsequent result that makes use of the thresholds in its calculation will also benefit from the findings brought by this paper, making the remained of Ceni's work overall more accurate, which was our initial goal.

8.3 Further steps

Despite bringing an improvement, the hereby proposed set of methods can be furtherly refined:

- As mentioned in Section 6.1, the chosen cluster-validation methods work by looking for compact clusters, and are not able to discern clusters with different shapes (e.g. circles or half moons). While the two methods work perfectly in the example proposed in Ceni's work, it is not necessarily true for every possible PDVs dataset, and therefore in some cases a different method should be used if the problem arises (e.g. methods based on graph cuts).
- Now that we have established that the representation of the points produced with PCA can and should be improved, we may want to use a different criterion to establish the number of dimensions for the reduction. An idea might be to cross validate using different PDVs dataset produced by different ESNs (which were not available at the time of writing) and use BSE on a validation set as error function to choose a generally correct number of dimensions.
- Even though the structure of the neural network used to reduce the dimensionality of the points has been accurately tested, it is possible that a better combination of structure, number of hidden units, activation functions, learning rate, optimiser and optimiser-specific hyper-parameters exists. Unfortunately, because of time constraints and computational power limitations of the machines used, not all such combinations could be tested, leaving some possible room for improvement if deemed necessary in further applications.
- The considerations above can also be applied to the proposed solution to the Out-of-Sample problem (an appositely trained neural network) and to the pre-images problem.
- In the sampling process two different possible improvements can be made:

1. The implementation of NNI is relatively efficient and capable of producing a large amount of new points in a short time, but an even more efficient implementation might exist.
2. At the moment it is possible to introduce noise in the calculation of new points with NNI. The addition of noise is just random Gaussian noise in the d coordinates of the low-dimensional points and does not take into account the local linearly of the manifold. In an improved version of the method the noise would be coplanar with the local patch of the manifold.

– As discussed in Section 4.5, BAE does not scale well with the number of dimensions of the input and for this reason BSE had to be developed, but the implementation proposed does not allow a relative measure of error, which would be desirable in many applications. A bettered version of BSE would be relative and in a predefined range.

References

[1] A. Ceni, P. Ashwin, and L. Livi, "Interpreting Recurrent Neural Networks Behaviour via Excitable Network Attractors," *Cognitive Computation*, Mar 2019.

[2] D. P. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.

[3] S. Grossberg, "Recurrent neural networks," *Scholarpedia*, vol. 8, no. 2, p. 1888, 2013.

[4] H. Jaeger, "Echo state network," *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[5] D. Castelvecchi, "Can we open the black box of AI?," *Nature News*, vol. 538, no. 7623, p. 20, 2016.

[6] P. Ashwin and C. Postlethwaite, "Designing heteroclinic and excitable networks in phase space using two populations of coupled cells," *Journal of Nonlinear Science*, vol. 26, no. 2, pp. 345–364, 2016.

[7] Y. Lu, I. Cohen, X. S. Zhou, and Q. Tian, "Feature selection using principal feature analysis," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 301–304, ACM, 2007.

[8] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative review," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[9] E. Golchin and K. Maghooli, "Overview of manifold learning and its application in medical data set," *International journal of Biomedical Engineering and Science*, 2014.

[10] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel PCA and de-noising in feature spaces," in *Advances in neural information processing systems*, pp. 536–542, 1999.

[11] B. Schölkopf, "The kernel trick for distances," in *Advances in neural information processing systems*, pp. 301–307, 2001.

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

[13] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[14] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[15] C. Doersch, "Tutorial on Variational Autoencoders," *CoRR*, vol. abs/1606.05908, 2016.

[16] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 12, pp. 1650–1654, 2002.

[17] T. M. Kodinariya and P. R. Makwana, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.

[18] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, pp. 849–856, 2002.

[19] H. Strange and R. Zwiggelaar, "A generalised solution to the out-of-sample extension problem in manifold learning.," in *AAAI*, pp. 293–296, 2011.

[20] A. Jansen, G. Sell, and V. Lyzinski, "Scalable out-of-sample extension of graph embeddings using deep neural networks," *Pattern Recognition Letters*, vol. 94, pp. 1 – 6, 2017.

[21] J. Frost, "How to Identify the Distribution of Your Data." `http://statisticsbyjim.com/hypothesis-testing/identify-distribution-data/`. Accessed: 2018-11-12.

[22] P. Zhang, Y. Ren, and B. Zhang, "A new embedding quality assessment method for manifold learning," *Neurocomputing*, vol. 97, pp. 251–266, 2012.

[23] AT & T Laboratories Cambridge, "Olivetti faces," 1992-1994.

[24] Y. LeChun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits."

[25] C. O. Sakar, G. Serbes, A. Gunduz, H. C. Tunc, H. Nizam, B. Sakar, M. Tutuncu, T. Aydin, M. Isenkul, and H. Apaydin, "A comparative analysis of speech signal processing algorithms for Parkinson's disease classification and the use of the tunable Q-factor wavelet transform," *Applied Soft Computing*, vol. 74, 10 2018.

[26] A. B.-H. G. D. Isabelle Guyon, Steve R. Gunn, "Result analysis of the NIPS 2003 feature selection challenge," 2004.

[27] S. C. J. M. B. J. D. K. R. J. Lyon, B. W. Stappers, "Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach," *Monthly Notices of the Royal Astronomical Society*, vol. 459 (1), pp. 1104–1123.

[28] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*, vol. 501. John Wiley & Sons, 2009.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[30] N. Ketkar, "Introduction to keras," in *Deep Learning with Python*, pp. 97–111, Springer, 2017.