Experimental study of Manifold learning and tangent propagation

by

Ayazhan Aman

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Master of Applied Mathematics

at the

NAZARBAYEV UNIVERSITY

Apr 2020

© Nazarbayev University 2020. All rights reserved.

Author .....
Department of Mathematics
May 4, 2020

Certified by .....
Rustem Takhanov
Associate Professor
Thesis Supervisor

Accepted by .....
Daniel Pugh
Dean, School of Science and Humanities

Experimental study of Manifold learning and tangent propagation

by

Ayazhan Aman

Submitted to the Department of Mathematics
on May 4, 2020, in partial fulfillment of the
requirements for the degree of
Master of Applied Mathematics

Abstract

In many data mining problems dealing with artificially high-dimensional data, can cause a lot of difficulties. This is due to the fact that interpreting a high-dimensional data can be very challenging. There are several ways to avoid that kind of difficulties and one of them is the dimension reduction method. Nowadays, manifold learning approach is widely used in various dimension reduction problems. In this work we propose a new technique of finding data manifold by using auto-encoders.

Thesis Supervisor: Rustem Takhanov
Title: Associate Professor

Acknowledgments

This thesis work finalizes my study on the master program at Nazarbayev University. First, I am extremely grateful to my thesis advisor, Associate Professor Rustem Takhanov, who guided me through the whole research process, for his kind responsiveness and patience. From the very beginning Professor provided an extensive support and gave a strong foundation knowledge. I am very grateful for his time and supportive help throughout the whole period of working on my thesis.

I am also grateful to each of the members of Thesis Committee, my second reader Professor Zhenisbek Assylbekov for their comments and suggestions needed for improvements of my thesis. I am also thankful to Professor Nurlan Ismailov for agreeing to be a thesis committee member and to review my work.

Contents

Chapter 1

Overview

1.1 Introduction

Data mining is focused on identifying hidden information and patterns from a huge amount of data. This information, which is supposed to be presented in the structure of the dataset, can be obtained using data analysis algorithms. There are several major issues that data mining deal with. They contain specific problems like outlier analysis, classification, regression, clustering, forecasting, etc. Such problems are very complicated for data mining since they play an important role in data mining applications.

Some dataset structures can be expressed in geometric, algebraic, and probabilistic viewpoints. In case when the real-world high-dimensional data is involved many efficient algorithms are based on the geometric approach. Apart from that, this algorithm is also useful when the occurrence of the "curse of dimensionality" causes difficulties in solving various problems.

In some data mining problems, we can observe an interesting phenomenon: although we are given data with many features, in whole future space data support could have in real sense low dimensionality. More specifically, the intrinsic dimension is formed from data with a high dimension that is a "tiny" part of high dimensional "observation space". A manifold model can be a good example of the geometric data model that characterizes the low-dimensional structure of the data space [3]. In the manifold model algorithm, given data points lie on some low-dimensional data manifold (DM). This data manifold is included in an external high-dimensional "observation space". Different data analysis problems have been studied suggesting that given data is manifold- valued. These problems are traditionally known as "manifold learning problems". [4, 5]. In particular, manifold learning problems focus on figuring out the data manifold's structure which has a low dimension in some high-dimensional space.

One way to extract data from the data space is called sampling models. Since data points are chosen randomly and independently from each other on some probability measure, and the support of this data matches with the data space, such models are said to be probabilistic.

1.2 Background

Definition. If for any point a in some topological space D there is a neighborhood O such that a mapping $g : O \longrightarrow \mathbb{R}^n$ is homeomorphic, then D is called *n-dimensional Manifold*. A pair $(O, g : O \longrightarrow \mathbb{R}^n)$ called a *chart*. [1]

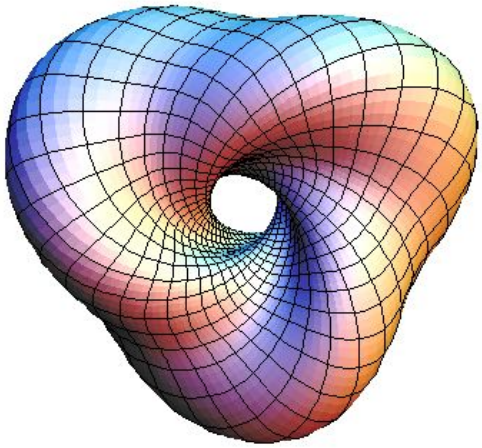


Figure 1-1: n-dimensional Manifold.

Let $\mathbb{R}^m$ be an m-dimensional space and $D \subseteq \mathbb{R}^m$, which is a "smooth-enough" k-dimensional data manifold, where $k \leq m$. Suppose that a dimension of manifold D was pre-calculated in advance. Assume also that there is a homeomorphism $g : B \longrightarrow D$ from $B \subset \mathbb{R}^k$ onto the manifold $D = g(B)$. Thus, our data manifold D is defined as $D = \{A = g(p) \in \mathbb{R}^m : p \in B \subset \mathbb{R}^k\}$, where B is an open set in $\mathbb{R}^k$. Since g is a homeomorphism, there exists an inverse function $f = g^{-1} : D \longrightarrow B$, which defines low-dimensional structure on data manifold D. [2]

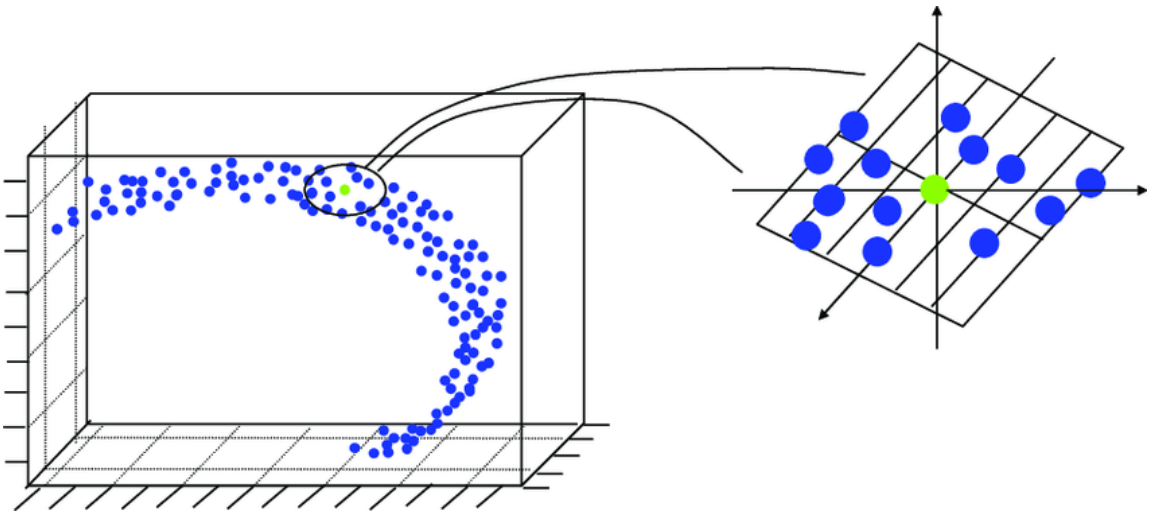


Figure 1-2: n-dimensional Manifold.

Definition. Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be a function and let the first-order partial derivatives of f exist on $\mathbb{R}^n$. Then we define the Jacobian matrix as:

$$\mathbb{J}_f(x) = [\frac{\partial f}{\partial x_1}(x) \ \ldots \ \frac{\partial f}{\partial x_n}(x)] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Hence,

$$span(J_f(x)) = span(\frac{\partial f}{\partial x_1}(x) \ \ldots \ \frac{\partial f}{\partial x_n}(x))$$

Definition. Suppose first-order derivatives of functions $f(A)$ and $g(p)$ exist and Jacobian matrices of them have a form $J_f(A) \subseteq \mathbb{R}^{k \times m}$ and $J_g(p) \subseteq \mathbb{R}^{m \times k}$ respectively. Then we define *Tangent space* to the data manifold D at point $A \in D$ as follows:

$$T(A) = Col(J_g(f(A)),$$

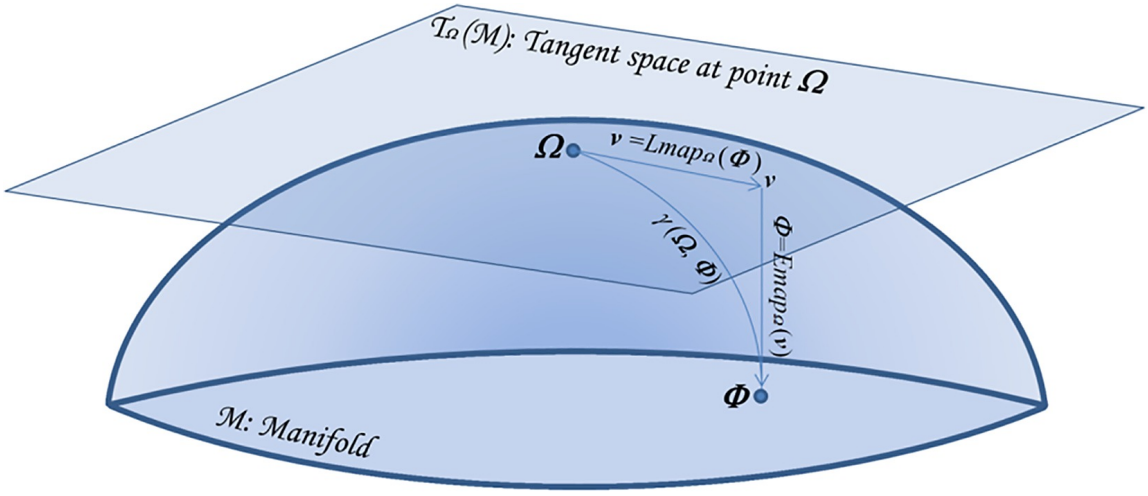
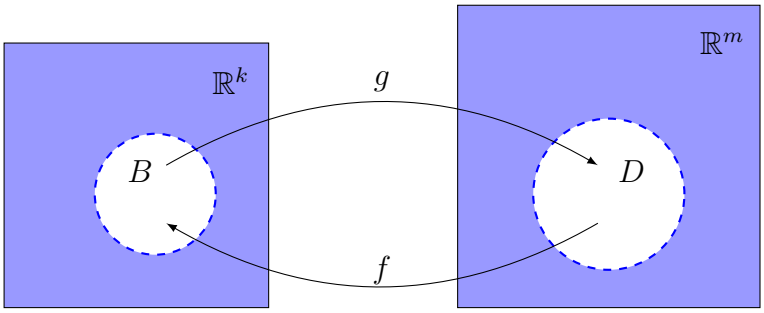


Figure 1-3: Tangent space to the Manifold.

where T(A) is a linear space in $\mathbb{R}^m$ with dimension k, if $\text{rank}(J_g(f(A)) = k$.



From the above formulations, we can observe the following identities: $g(f(A)) \equiv A$ and $f(g(p)) \equiv p$. To be more clear, the mapping $g : B \longrightarrow D$ is called *embedding mapping* and $f : D \longrightarrow B$ is *recovery mapping*.

By the machine learning perspective, these "embedding" and "recovery" mappings can be easily determined by the so-called auto-encoders. The principle of

auto-encoders is to map given high-dimensional data into low dimensional space and then uncompress them into original space such that the error will be very small.

1.3 Manifold Learning

Let D be a Data Manifold. Suppose sample size N is large enough. In other words, this Data Manifold is "well-sampled". Also, let us assume haphazardly and independently from each other sampled points of dataset

$$A_N = \{A_1, A_2, \ldots, A_N\}$$

which are taken from the manifold. They are sampled from Data Manifold D subject to some probability measure ν.

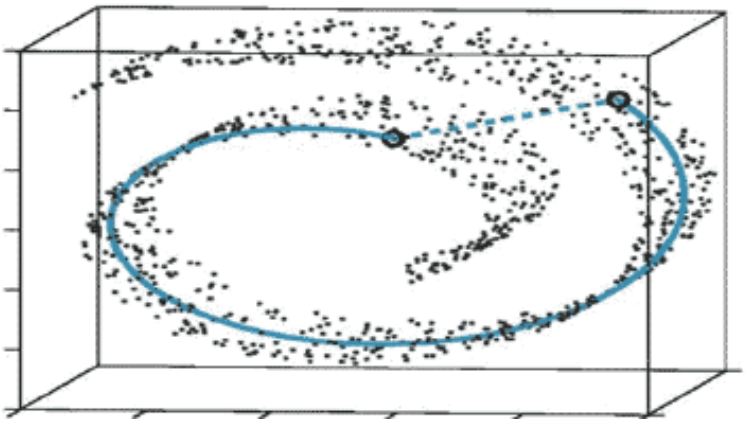


Figure 1-4: An example of data manifold.

Manifold learning is one of the principal techniques of dimensionality reduction problems. The main task of manifold learning is to describe the characterization of the low-dimensional parameterization of some $k-$ dimensional Data Manifold D which is randomly sampled from dataset A_N.

From the perspective of data mining or machine learning the manifold learning problem becomes the manifold embedding problem. To be more precise, we consider a dataset A_N to formulate a low-dimensional structure of the Data manifold D.

$$g : A \in D \subset \mathbb{R}^m \to y = g(A) \in Y_g = g(D) \subset \mathbb{R}^k$$

where g is called embedding mapping from the Data manifold D to Y_g. Y_g is called feature space which keeps some particular data manifold's topological and geometric properties. There is a wide range of examples of manifold embedding techniques, namely, linear embedding, ISOMAP, Hessian and Laplacian eigenmaps, etc.

Generally, In many kinds of data mining or machine learning problems manifold embedding is considered as a basic stage. Throughout the entire procedural way of learning, it is preferable to use reduced features $y = g(A)$ from the feature space in preference to using initial m-dimensional vectors A. As mentioned before, that mapping should keep topological and geometrical properties of the data manifold. But this can be insufficient since it can lead to loss of essential data. So allowing the substitution of primitive vector A by reduced vector $y = g(A)$ can cause gaps in the initial information. To avoid this kind of loss, the abstract function g must store as much information as possible from the source data, which is part of a multi-dimensional observation space. Precisely speaking, the goal is to achieve the return

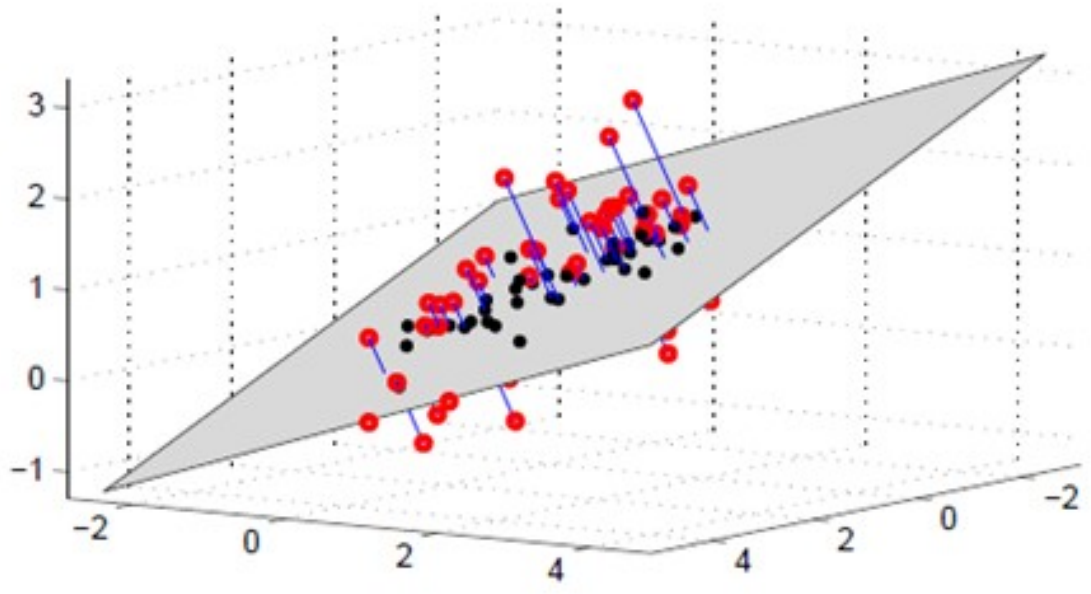


Figure 1-5: Data lie on low-dimensional manifold.

of multidimensional points A from reduced features $y = g(A)$ which exactly corresponds to their low dimensional projection, so that the error of recovery should be unessential. This will provide a way of keeping the initial information included in high-dimensional data. Consequently, there is a necessity to find a new mapping

$$f : y \in Y_g \rightarrow A = f(y) \in \mathbb{R}^m$$

where f is called recovery mapping from the feature space Y_g to the external space R^m. Embedding mapping in conjunction with recovery mapping provides such relation

$$r_{g,f}(A) \equiv f(g(A)) \approx A, \qquad \forall A \in D$$

where $r_{g,f}$ is consequent of sequential application of embedding and recovery mappings to a vector $A \in D$.

There is a possibility to assess of optimality embedding and recovery (g, f) at a point $A \in D$ by new notion called the reconstruction error and denote it as $\delta_{g,f}(A) = |A - r_{g,f}(A)|$. Moreover, it is possible to define a k-dimensional data manifold $D_{g,f} = \{A = f(y) \in \mathbb{R}^m : y \in Y_g \subset \mathbb{R}^k\}$ enclosed in $\mathbb{R}^m$ and characterized by an individual chart which is on the feature space Y_g. Furthermore, it is also possible to estimate manifold proximity by such inequality below

$$d_H(D_{g,f}, D) \leq sup_{A \in D} |r_{g,f}(A) - A|$$

$$D \approx D_{g,f} \equiv r_{g,f}(D)$$

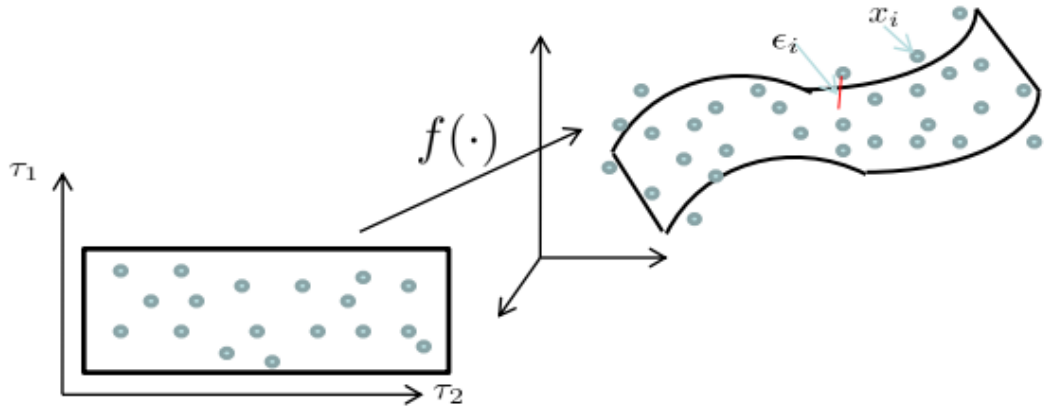


Figure 1-6: Manifold learning.

There are several ways to determine the data manifold D from the feature space Y_g. One of such methods is a principal component analysis (PCA). But this technique is applicable only if our manifold is linear. What if we have a nonlinear manifold? In that case, we can use a specific type of artificial neural network called auto-encoders.

1.4 Autoencoders

Autoencoders are a particular type of Artificial Neural Networks which tries to reproduce its input. Generally speaking, the network learns the way how to efficiently copy the training data, i.e. we return to the same space from which we started. Firstly, they construct a code that reduces input data dimensions and then uncompress that code back to reproduction which is almost the same as the original input.

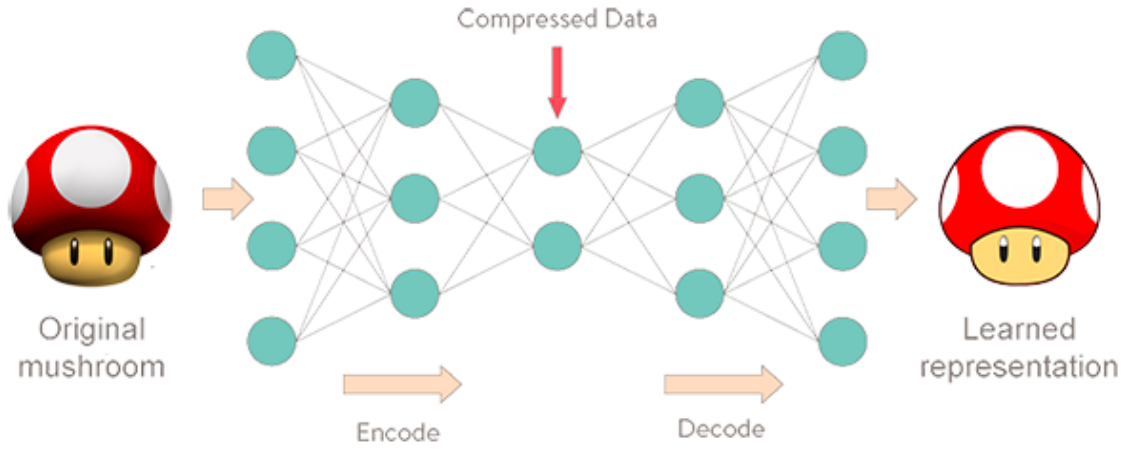


Figure 1-7: An example of how autoencoder works.

An autoencoder can be viewed as consisting of three main components: an *encoder* which compresses data from the input layer and generates a *code* which represents the input; and a *decoder* which produces output by applying exclusively this code. In mathematical terms, there are two functions: the encoder function $h = f(x)$ (mapping input x to some low-dimensional space or code h), and the decoder function $r = g(h)$ (mapping through h to output r, which is a reconstruction of input).

There is one important thing, dimensions of input and output must be equal. Therefore, the main point of autoencoding is this: we have an initial m-dimensional space, there is an intermediate k-dimensional space where $k < m$, and the final space has the same dimension as the first one. That is, we need to project the original vectors into a low-dimensional space, then return to the high-dimensional original space and at the same time make our output almost like input data. The process of mapping from an input space to a low-dimensional one is *encoding*, and the reverse is *decoding*. The standard choice of g encoding and f decoding functions, that is, the simplest auto-encoder is when $g(a) = \sigma(Wa+c)$ and $f(y) = \sigma'(W'y+c')$ are single-layer neural networks. g and h called *decoder* and *encoder*, respectively. Figure 1-8 illustrates the architecture of autoencoders.

$$g : \mathbb{R}^m \to \mathbb{R}^k$$

$$f : \mathbb{R}^k \to \mathbb{R}^m$$

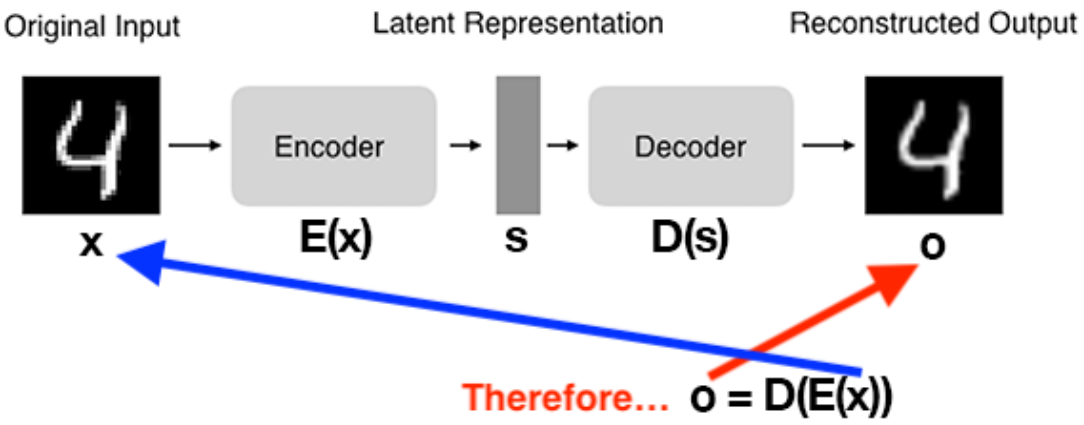


Figure 1-8: Architecture of an autoencoder.

Let's take a closer look at the structure of building autoencoders. Both of the components encoder and decoder are fully-connected artificial neural networks. Their hyperparameters must be determined in advance. We can set the number of layers as much as we want. A number of nodes in every next layer should be decreased in the encoder and increased in the decoder, conversely. We emphasize that structure of decoder is a mirror reflection of the encoder. They must be exactly symmetrical to each other. However, code is a single-layer artificial neural network with an arbitrary number of nodes. The visualization of the architecture of the autoencoder is illustrated in Figure 1-9.

Since the goal of autoencoders is to reproduce input as identical as possible, we need a loss function to measure how accurately the output resembles the input. We can either use binary cross-entropy if values of input lie between 0 and 1, or mean squared error, otherwise. Ultimately, if we are given a set of data $x_1, x_2, \ldots, x_N \in \mathbb{R}^m$ our goal is to find such encoder function $h = f(x)$ and decoder function $r = g(h)$ so that $g(f(x_i)) \approx x_i$, or to minimize

$$f, g = argmin_{f,g} \sum_{i=1}^{N} ||x_i - g(f(x_i))||^2$$

The question arises, why do we do this? It turns out that having solved this problem, we actually proposed a solution to the manifold learning problem. We will look at a more detailed explanation in the next Section.

1.5 Manifold Learning and Autoencoders

Let m is given. We can choose the dimension of the intermediate space d ourselves. Let's assume that we have trained our autoencoder, i.e. g and f single-layer neural

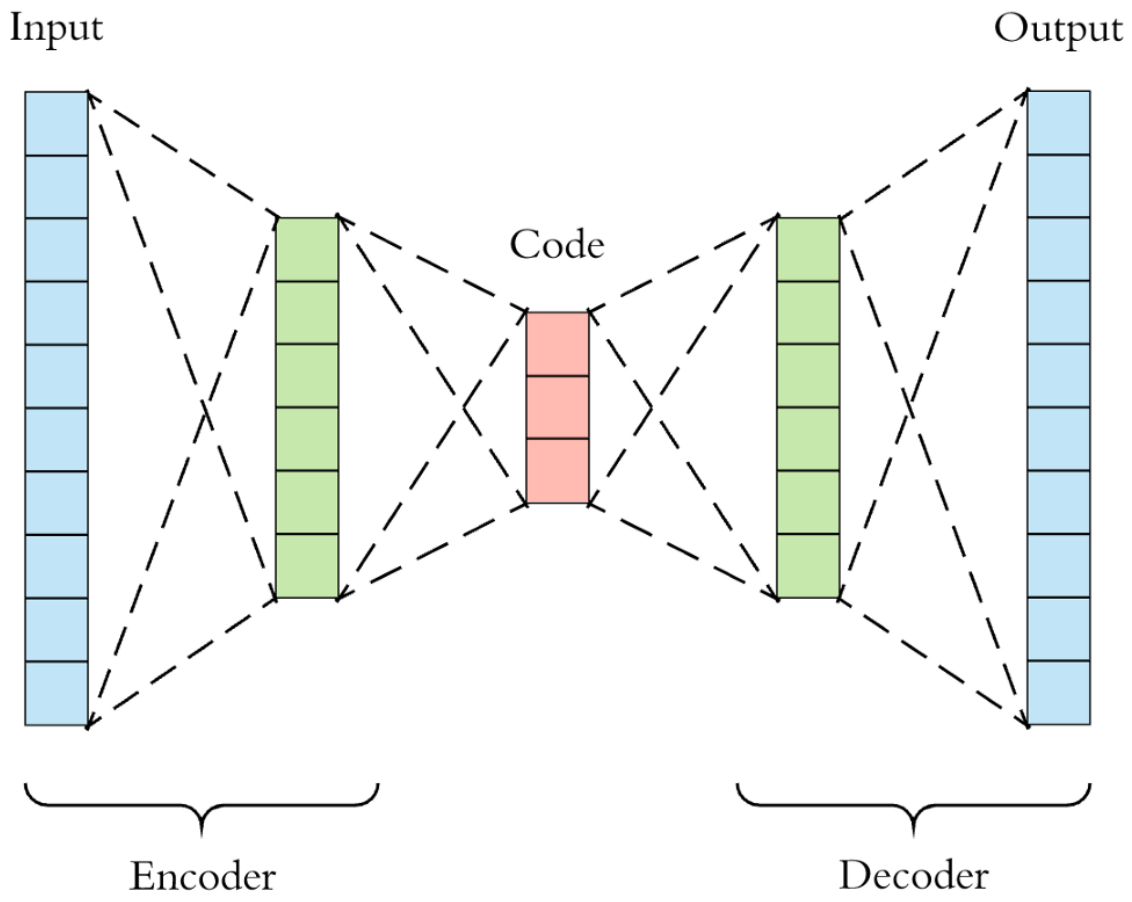


Figure 1-9: Scheme of basic autoencoder.

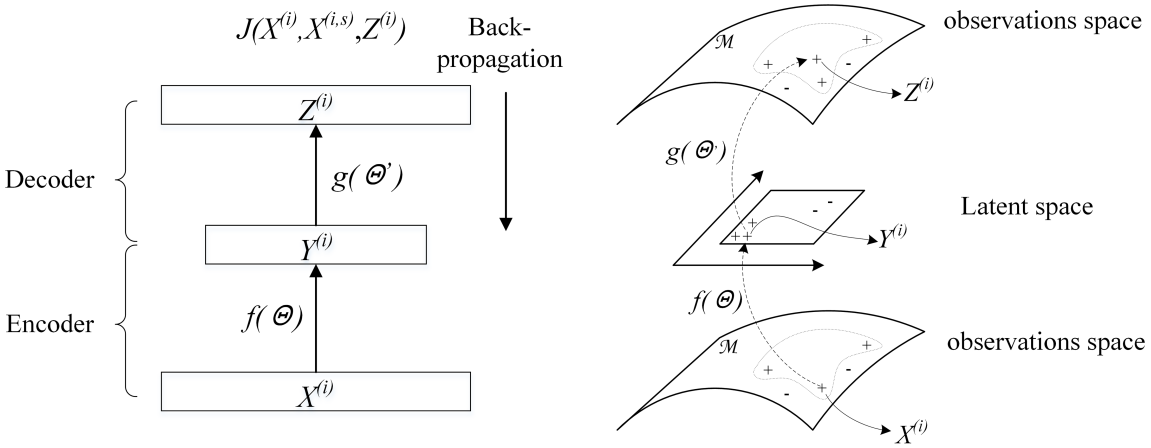
networks whose parameters are trained. Thus, we have two mappings: encoding and decoding, which result in small losses. What will be the image of this mapping?

Let us have m dimensional space $\mathbb{R}^m$, there is also an intermediate space $\mathbb{R}^k$ which is a code space, and we return at the end to $\mathbb{R}^m$. More precisely, we have the initial point a_i, it goes into the image $g(a_i)$, then we map it all in $f(g(a_i))$. But $g(a_i)$ is k dimensional space. In fact, our image will always be in $f(\mathbb{R}^k)$, i.e. roughly speaking, there is a k-dimensional space and its mapping to $\mathbb{R}^m$ will be a subset of that space. In other words, the image of autoencoders does not lie in $\mathbb{R}^m$, but in a certain subset

of $\mathbb{R}^m$. $f(\mathbb{R}^k)$ is an image of the intermediate space $\mathbb{R}^k$, namely, it is an image of the code space. Since f is a single layer neural network it is an injective function. Hence, the mapping that the autoencoder builds from a low dimensional to a high dimensional space is always injective.

An important point, after we trained the auto-encoder, our vectors a_i will not be in the original space $\mathbb{R}^m$, but in some subset of $\mathbb{R}^m$, which is the image of $\mathbb{R}^k$ with respect to an injective, continuous, smooth mapping. Since f is a neuron, it will be an infinite differentiable smooth mapping. Thus after encoding and decoding the image of a_i will lie on k dimensional manifold.

In the idea of auto-encoders, even though this is an applied thing, we can see an analogy with the manifold learning problem. In fact, we actually project our initial points onto a k- dimensional manifold.



Let us assume that we are given points $a_1, a_2, \ldots, a_N \in \mathbb{R}^m$ and a natural number k which is much less than m ($k << m$). In view of the fact that N is large enough,

interpreting such data can complicate the process. Therefore, in order to simplify the task, let us suppose that given data concentrates around a nested low-dimensional manifold in a multidimensional space. Hence, this implies the necessity to define a k-manifold $D \subseteq \mathbb{R}^m$ such that $a_1, a_2, \ldots, a_N \in D$, i.e.: points will lie on this embedded k-manifold D. We can give a more precise formulation.

We are given points $a_1, a_2, \ldots, a_N \in \mathbb{R}^m$ and a natural number ($k << m$). We need to find such points $b_1, b_2, \ldots, b_N \in \mathbb{R}^m$ called *"substitutes"* so that

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||a_i - b_i||^2} < \epsilon,$$

and a k-manifold $D \subseteq \mathbb{R}^m$ such that $b_1, b_2, \ldots, b_N \in D$.

Suppose that these "substitutes" can be obtained by mapping $b_i = f(g(a_i))$ where

$$g : A \in D \subset \mathbb{R}^m \to y = g(A) \in Y_g = g(D) \subset \mathbb{R}^k$$

is embedding mapping from the data manifold to the feature space and

$$f : y \in Y_g \to A = f(y) \in \mathbb{R}^m$$

is recovery mapping which maps points from feature space to the ambient space $\mathbb{R}^m$ as it was discussed before in previous sections. Also assume that f is one-to-one function and D is k-dimensional manifold determined as

$$D = f(\mathbb{R}^k) = \{f(y) | y \in \mathbb{R}^k\}.$$

Then

$$a_i \approx b_i = f(g(a_i)) \in f(\mathbb{R}^k) = D$$

Thus, our goal is to find "substitutes" $\forall i, b_i \in D$ and minimize

$$\frac{1}{N}\sum_{i=1}^{N} ||a_i - b_i||^2.$$

Summarizing all of the above it follows that we built a mapping $\phi = f \circ g$ by which our given data concentrates around low-dimensional k-manifold D. If we take a closer look, then this function is nothing more than auto-encoders. In fact, b_i - output of the auto encoder is the replacement of our initial points a_i. Now we know that all b_i lie on a certain manifold D which is an image of $\mathbb{R}^k$ at f. Our manifold is an image of $\mathbb{R}^k$, and b_i-substitutes is nothing more than an image of $g(a_i)$ with respect to auto-encoding. The output of the auto-encoder will lie on a certain subset of $\mathbb{R}^m$ that is locally k-dimensional. $f(g(\mathbb{R}^m))$ is the space onto which we projected our original vector. In conclusion, we can say that we can set our "substitutes" using auto-encoders, where "embedding" and "recovery" mappings are "encoder" $g(a) = \sigma(Wa + c)$ and "decoder" $f(y) = \sigma'(W'y + c')$, respectively. Auto-encoding is already a manifold learning.

Chapter 2

Main

2.1 Classic method

In the previous section, we showed that auto-encoders are analogous to solving our manifold learning problem. The question is, how do we find the best auto encoder?

Let us look at the most classic method of finding auto-encoders. We are given points $a_1, a_2, \ldots, a_d \in \mathbb{R}^m$ and a natural number $k << m$. Let us denote points $b_1, b_2, \ldots, b_d \in \mathbb{R}^m$ as "substitutes", which are the output of the auto-encoder and the reconstruction of our initial points $a_1, a_2, \ldots, a_d \in \mathbb{R}^m$. We have an auto-encoder in the form of

$$\phi(\overline{a}) = f(g(\overline{a})),$$

which is the composition of our encoder and decoder functions. This is a smooth mapping $\phi : \mathbb{R}^m \to D$ from a high-dimensional space $\mathbb{R}^m$ to a k-dimensional manifold

D. That is, $b_i = f(g(a_i))$. Let us denote

$$I(f,g) = \sum_{i=1}^{d} ||a_i - f(g(a_i))||.$$

Our task is to find such f^* and g^*, so that

$$f^*, g^* = argmin_{f,g} I(f,g)$$

2.2 Concurrent method

Let us assume that we built the auto-encoder in the classical way described above. In other words, the function $\phi(\overline{a})$ actually displays our data from $\mathbb{R}^m$ to $\mathbb{R}^m$. But in reality, the image from $\mathbb{R}^m$ is a subset of the image $f(\mathbb{R}^k)$. In fact, $\phi(\overline{a})$ maps $\mathbb{R}^m$ into a certain manifold D which is k-dimensional.

Upon the fact of auto-encoding, we have already compressed our data until k and decompressed it back. The idea is that why don't we compress our data more strongly. What if we don't want to stop at k, and we want to map the data to even less space. $f(\mathbb{R}^k)$ is a k-dimensional thing and $\phi(\overline{a}) \subseteq f(\mathbb{R}^k)$. Why not make this image $\phi(\overline{a})$ even smaller than $f(\mathbb{R}^k)$. And how to do it?

Our new approach is based on the following theorem:

Theorem 1. *Let $\phi : \mathbb{R}^m \to \mathcal{D}$ be a smooth mapping such that $\mathcal{D} \subseteq \mathbb{R}^m$ is a j-dimensional manifold. Then for any point $\mathbf{a} \in \mathbb{R}^m$, we have* $\text{rank}(\frac{\partial \phi}{\partial \mathbf{a}}) \leq j$.

Namely, the image of the auto-encoder ϕ, which implements a function from $\mathbb{R}^m$

to $\mathbb{R}^m$, is actually not just a subset of $\mathbb{R}^m$, but a k-dimensional manifold $\mathcal{D}$. We assume that j is even smaller than k ($j < k$), where k is the dimension of the code space.

It is easy to show that if $\mathcal{D}$ is the image of some open set in $\mathbb{R}^j$ and if we have a map from $\mathbb{R}^m$ to this image $i(\Omega)$ where i is a map from Ω to $\mathbb{R}^m$, $i : \Omega \to \mathbb{R}^m$ is injective, then you can notice that $\phi : \mathbb{R}^m \to i(\Omega)$. Since $i : \Omega \to \mathbb{R}^m$, then inverse of i is $i^{-1}(\phi(\mathbf{a})) : \mathbb{R}^m \to \mathbb{R}^j$. The mapping of the inverse of i is from $\mathbb{R}^m$ to $\mathbb{R}^j$. We use the simple fact, the product of the Jacobian of the inverse mapping and Jacobian of ϕ since this mapping is from $\mathbb{R}^m$ to $\mathbb{R}^j$. Then $\text{rank}(\frac{\partial i^{-1}}{\partial \phi} \cdot \frac{\partial \phi}{\partial \mathbf{a}}) \leq j$. $\frac{\partial i^{-1}}{\partial \phi}$ is full rank matrix. And that means that $\text{rank}(\frac{\partial \phi}{\partial \mathbf{a}}) \leq j$.

Earlier in the previous subsection, we described a standard auto-encoder. Now our task is to build such an auto-encoder, where for each point of this auto-encoder the rank of the Jacobian matrix will be less than or equal to j. To be more precise, our task is to construct such a mapping from $\mathbb{R}^m$ to $\mathbb{R}^m$ which has the following property: ϕ is a superposition of the functions f and g - as we assumed in auto-encoders, $\phi = f \circ g$, $g : \mathbb{R}^m \to \mathbb{R}^k$, $f : \mathbb{R}^k \to \mathbb{R}^m$, and we add an additional constraint. We want our image to be a j-dimensional manifold. In the architecture of auto-encoding itself, we have already noted that our image will be at least k-dimensional. But we want to further reduce the dimension. We want to add the condition that the rank of Jacobian is less than or equal to j, where $j < k$. We impose an additional condition. We want the auto-encoder image to be even lower-dimensional. To do this, we needed constraints.

Concluding all of the above, our goal will be now to minimize:

$$I(f,g) = \sum_{i=1}^{d} ||a_i - f(g(a_i))||^2$$

over f, g (as in classical algorithm) and, simultaneously, to force the constraint:

$$\text{rank}\frac{\partial f(g(a_i))}{\partial a} \leq j$$

for every data point a_i.

Now the question is how to solve this problem?

Let us assume that we already have some approximate solution to this problem. Suppose that we have at iteration some approximate value of f_n and g_n. The idea is that we build f_{n+1} and g_{n+1} that will be even better than the previous ones. But how to build it? To do this, we need to minimize our functional $I(f,g)$, but we also impose an additional constraint. We want the Jacobian at each point in our data to be the same as the Jacobian of the previous pair of f_n and g_n. In other words, we minimize the difference of Jacobian at points on the previous and next steps. In general, let us say we have an auto-encoder in the previous step. Our goal is to build a new auto-encoder at a new step, which will also minimize our functional $I(f,g)$, but in addition we want the Jacobian corresponding to the auto-encoder to be the same as in the previous step. Roughly speaking, we do SVD truncation to the j-th member of the Jacobian in the previous step. And then we will definitely sum it all up for all our points.

In other words, the idea is to minimize $I(f, g)$ and find a new auto-encoder f_{n+1} and g_{n+1}. It turns out a new auto-encoder is searched for from two conditions:

- We minimize our functional $I(f, g)$ as in the classical algorithm.
- Adding a new term, the essence of which is that the Jacobian of the new autocoder at each point must be the same as the Jacobian at the previous step. Just not the full Jacobian, but the SVD of Jacobian.

2.3 New approach

We will apply the following strategy to achieve the latter two goals.

1. At every iteration n, we calculate the autoencoder f_n, g_n.
2. Given f_{n-1}, g_{n-1}, we calculate new f_n, g_n by a) minimizing $I(f, g)$ (as in classical algorithm), b) adapting the jacobian $\frac{\partial f(g(\mathbf{x}_i))}{\partial \mathbf{x}}$ to $SVD_j(\frac{\partial f_{n-1}(g_{n-1}(\mathbf{x}_i))}{\partial \mathbf{x}})$, where $SVD_j(A)$ is a truncation of $SVD(A)$ at j-th term

I.e. at step n we minimize:

$$f_n, g_n = \arg\min_{f,g} I(f, g) + \lambda \sum_{i=1}^{N} |\frac{\partial f(g(\mathbf{x}_i))}{\partial \mathbf{x}} - SVD_j(\frac{\partial f_{n-1}(g_{n-1}(\mathbf{x}_i))}{\partial \mathbf{x}})|^2$$

Since $SVD_j(\frac{\partial f_{n-1}(g_{n-1}(\mathbf{x}_i))}{\partial \mathbf{x}})$ is of rank j, then after many iterations we will converge to f_N, g_N such that:

$$\text{rank}\frac{\partial f_N(g_N(\mathbf{x}_i))}{\partial \mathbf{x}} \leq j$$

2.4 New algorithm with details

Our algorithm is the following (we have to choose a parameter λ first).

Algorithm 1 The alternating scheme

$P_0 \longleftarrow \mathbf{0}, f_0, g_0 \longleftarrow \mathbf{0}$
for $t = 1, \cdots, T$ **do**
  $f_t, g_t \longleftarrow \arg\min_{f,g} I(f,g) + \lambda \sum_{i=1}^{N} |\frac{\partial f(g(\mathbf{x}_i))}{\partial \mathbf{x}} - P_{t-1}^{i} \frac{\partial f_{t-1}(g_{t-1}(\mathbf{x}_i))}{\partial \mathbf{x}}|^2$
  for $k = 1, \cdots, N$ **do**
    Calculate $M_t^k = \left[\langle \frac{\partial f_t(g_t(\mathbf{x}_k))}{\partial x_i}, \frac{\partial f_t(g_t(\mathbf{x}_k))}{\partial x_j} \rangle \right]$
    Find $\{\mathbf{v}_i\}_1^n$ s.t. $M_t^k \mathbf{v}_i = \lambda_i \mathbf{v}_i$, $\lambda_1 \geq \cdots \geq \lambda_n$ and set $P_t^k \longleftarrow \sum_{i=1}^{d'} \mathbf{v}_i \mathbf{v}_i^T$
  end for
end for
Output: $P_T^k, k = 1, \cdots, N$

Chapter 3

Experiments

3.1 Experimental setup

In this work, several experiments were carried out to analyze the effectiveness and evaluate performance of our new algorithm using various datasets. All experiments were conducted on Python 3 programming language using TensorFlow software library.

The MNIST dataset was selected for the first tests. It is a large database of handwritten digits, where each image is black and white image of 28x28 pixels. This dataset is freely available and consists of 80,000 images including 60,000 training examples and 10,000 testing examples. The are 10 classes of images: digits from 0 to 9. The learning task is to predict the digit contained in the images.

For the classification we use a k-NN algorithm and compare accuracy. The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classifi-

cation and regression. In both cases, the input consists of the k closest training examples in the feature space. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

The k-NN is trained on the raw input vector using the Euclidean distance. We also use K-layer+k-NN in order to investigate, which of methods will lead us to better classification performance on these problems. The main difference from the usual k-NN is that in the K-layer + KNN, instead of the Euclidean distance, we calculate the distance in the code space.

We conducted experiments with the algorithm 1 that we applied for unsupervised data in database MNIST. Following [7], after training an autoencoder on the training data, we used the reduced encoding of an input point as a new object representation. Given that representation, we subsequently exploited it for classification of the test set data by KNN algorithm. The accuracy of classification on the test set shows the quality of the obtained representation, or the usefulness of our manifold learning technique. Alternative representation was obtained using the method of Manifold Tangent Classifier [7]. Results are reported below on the table.

	KNN	1-Layer RRJ+KNN	2-Layer RRJ	1-Layer MTC	2-Layer MTC
MNIST	89.92	**92.35**	**92.08**	90.55	91.15

Table 3.1: Results

Chapter 4

Conclusion

4.1 Conclusion

The idea of using autoencoders for manifold learning task is a part of data science folklore, but most of its realizations deal with the gradient descent as the standard tool for optimization of an objective function. We used Constant Rank Theorem to substantiate the use of an additional term that forces the Jacobian of an autoencoder to be of bounded rank. To minimize the penalized objective we developed a new algorithm based on alternating the gradient descent and an update of tangent spaces (or, to be more specific, a projection operators onto those spaces) to a learned manifold at data points. Our experiments demonstrate that the designed algorithm successfully minimizes the objective and solves the manifold learning task. The main disadvantage of our algorithm is its slow convergence for large databases and high dimensions of data (e.g. for CIFAR10). Future research will be dedicated to the question how an early termination and an optimization of parameters of the algorithm can be used to faster obtain a manifold without loss of desirable properties.

Bibliography

[1] Loring, W. Tu. "An introduction to manifolds." (2008), Springer

[2] A. P. Kuleshov, A. V. Bernstein, Yu. A. Yanovich, Manifold learning based on kernel density estimation, Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki, 2018, Volume 160, Book 2, 327–338

[3] Seung H.S. Cognition: The manifold ways of perception. Science, 2000, vol. 290, no. 5500, pp. 2268–2269. doi: 10.1126/science.290.5500.2268.

[4] Huo X., Ni X.S., Smith A.K. A survey of manifold-based learning methods. In: Recent Advances in Data Mining of Enterprise Data: Algorithms and Applications. Singapore, World Sci., 2008, pp. 691–745. doi: 10.1142/97898127798610015.

[5] Ma Y., Fu Y. Manifold Learning Theory and Applications. London, CRC Press, 2011. 314 p.

[6] Zhang, Tianhao, et al. "Linear local tangent space alignment and application to face recognition." Neurocomputing 70.7-9 (2007): 1547-1553.

[7] Rifai, Salah, Yann N. Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. "The manifold tangent classifier." In Advances in neural information processing systems, pp. 2294-2302. 2011.