

Incorporating Wasserstein metric in Autoencoder Training for Effective Image Compression

by

Antonina Rudakova

A research paper
presented to the University of Waterloo
in fulfillment of the
paper requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2024

© Antonina Rudakova 2024

Reader

The following served as readers of this report.

Reader: Hans De Sterck
Professor, Dept. of Applied Mathematics, University of Waterloo

Supervisor(s): Stephen Vavasis
Professor, Dept. of Combinatorics & Optimization,
University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this research paper. This is a true copy of the research paper, including any required final revisions, as accepted by my reareds.

I understand that my research paper may be made electronically available to the public.

Abstract

This study aims to improve autoencoder training methodology by incorporating the Wasserstein distance, the metric for capturing differences between image distributions, with the primary goal of integrating it into the autoencoder training process. To achieve this, our work presents an implementation designed to efficiently compute the entropy-regularized Wasserstein distance and its gradient between images. The numerical outcomes underscore the impact of this integration, revealing notable improvements in discrimination between input and reconstructed images. For instance, with an autoencoder configured at offset = 0.05 and epsilon = 0.1, the Wasserstein metric exhibited a substantial change of -0.11 when comparing various images of zeros, while the 2-norm showed a metric value of 0. Similarly, comparing images of 0 to 2 yielded a Wasserstein metric value of 1.49, outperforming the 2-norm's value of 0.08. Beyond individual pairs, our approach consistently resulted in enhanced discrimination, promising implications for the broader field of image classification. These results reinforce the Wasserstein distance's potential to optimize autoencoder-based image compression, contributing to the ongoing pursuit of employing advanced metrics for more efficient deep learning models in image processing tasks.

Acknowledgements

Here is an updated version (though in my opinion): I would like to express my heartfelt gratitude to my supervisor, Professor Stephen Vavasis, for his continuous guidance and unwavering support throughout my Master's studies. His mentorship has been invaluable, and I genuinely appreciate his shared knowledge and insight.

A special acknowledgment goes to my colleagues in the Computational Mathematics lab, who supported me during my academic journey and played a vital role in the discussions related to this thesis. Their collaborative spirit and encouragement have significantly enriched my research experience.

I am also sincerely thankful to the University of Waterloo for organizing the summer program, particularly in response to Russia's invasion of Ukraine. The university's support and commitment to the academic growth of Ukrainian students, especially during challenging times, have been instrumental. I am grateful for the opportunities provided by the University of Waterloo to foster learning and cultural exchange.

Once again, thank you to everyone who has supported me throughout my academic journey at the University of Waterloo.

Dedication

I dedicate this thesis to the unconditional support and love of my mother. Her encouragement and belief in my abilities have been a driving force throughout my academic journey.

To my close friends, who stood by me during the challenges and joys of this academic pursuit, your friendship has been a source of strength. Your understanding and encouragement have significantly impacted me, and I am fortunate to have you in my life.

A special mention goes to my best friends, Oleh and Diana. Your leadership, guidance, and exemplary dedication to your pursuits have inspired me to strive for excellence. Your friendship has been a guiding light, and I am thankful for your positive influence on my academic and personal growth.

Thank you for being an integral part of this journey.

Table of Contents

Reader	ii
Author's Declaration	iii
Abstract	iv
Acknowledgements	v
Dedication	vi
List of Figures	ix
List of Tables	x
1 Introduction	1
2 Wasserstein distance	3
2.1 Kantorovich Optimal Transport Problem (OTP)	4
2.2 Entropic Regularization of Optimal Transport	5
2.3 Sinkhorn's Algorithm	6
3 Autoencoder	9
3.1 Structure of Autoencoder	9

3.2	Autoencoder vs SVD	10
3.3	Loss function: Wasserstein distance	14
3.4	Wasserstein distance: Autodifferentiation	15
4	Observations	21
	References	26

List of Figures

2.1	Three densities p_1, p_2, p_3 [6].	3
3.1	An $A(n; m; n)$ Autoencoder Architecture [1].	10
3.2	Linear SVD with rank 2.	13
3.3	Affine SVD with rank 2.	13
3.4	Autoencoder with SGD and Adam optimizers.	14
4.1	Multidimensional scaling of Wasserstein distance and 2-norm	23
4.2	Annotated figures of Wasserstein distance and 2-norm.	25

List of Tables

3.1	Table of Figures	15
3.2	Truncation Error of the Algorithm 2	20
4.1	Algorithm 1 experiment with various parameter values.	21

Chapter 1

Introduction

Within the fields of artificial intelligence and image processing, researchers have been exploring novel strategies in an effort to achieve optimal picture compression while maintaining important characteristics. Neural networks' autoencoder class has shown to be a potent tool in this endeavour. These neural networks are made to learn compact representations by encoding incoming data and then reconstructing it. Improving autoencoder training methodologies becomes necessary as the necessity for effective picture compression increases.

Image compression is essential for many uses, from rapid transmission via communication systems to effective storage. Picture compression aims to present visual data more compactly while maintaining critical information. By learning compact representations of input data, a type of artificial neural networks called autoencoders has demonstrated promise in picture compression challenges ([14], [15], [18]). However, addressing issues with the accuracy of reconstructed pictures and the quality of learnt representations is necessary for improving autoencoders for efficient compression.

Characterized by an encoder-decoder design, autoencoders are models for unsupervised machine learning. The input data, which is typically an image, is mapped by the encoder into a latent space of lower dimension. The decoder then uses this reduced representation to reconstitute the original input. The key to autoencoders is their capacity to identify relevant characteristics throughout the encoding-decoding process, which allows them to learn effective data representations.

Image compression has shown to be a particularly successful use of this encoding-decoding technique. Autoencoders enable the construction of compact representations

that preserve crucial information by minimizing the dimensionality of the input data. This makes it possible to store and transmit pictures efficiently.

But conventional autoencoder training techniques might not necessarily produce the best outcomes, particularly when trying to achieve particular desirable features in the compressed form. This leads to the investigation of sophisticated criteria, such the Wasserstein metric, to direct the training procedure in order to get better compression efficiency.

In this article, we explore the incorporation of the Wasserstein metric into the training process of autoencoders to enhance its performance in image compression. An indicator of how different probability distributions are from one another is the Wasserstein metric, also referred to as the Earth Mover's Distance. Our goal is to enhance the reconstructed picture quality and encourage the acquisition of more significant image representations by utilizing the Wasserstein measure throughout the autoencoder training process.

The goal of this research is to incorporate the Wasserstein metric into the autoencoder's loss function, directing the training process to produce compressed representations consistent with the metric's focus on structure preservation. Through empirical evaluations and experiments, we argue that our Wasserstein distance algorithm is better in clusterization than the Euclidean norm, which provides promises to achieve better picture compression by incorporating Wasserstein distance algorithm as a loss function of an autoencoder.

Chapter 2

Wasserstein distance

Wasserstein distance is a distance function defined between probability distributions on a given metric space M . It is named after Leonid Vaseršteĭn. Intuitively, if each distribution is viewed as a unit amount of earth (soil) piled on M , the metric is the minimum "cost" of turning one pile into the other, which is assumed to be the amount of earth that needs to be moved times the mean distance it has to be moved. This problem was first formalised by Gaspard Monge in 1781 [9]. Because of this analogy, the metric is known in computer science as the earth mover's distance. the material in this chapter is based on the survey paper [13].

To see the difference between the Wasserstein distance and other distances consider figure 2.1 with three densities p_1, p_2, p_3 . Since $\int |p_1 - p_2| = \int |p_2 - p_3|$, each pair has the same distance in L1, L2, Hellinger etc. But Wasserstein distance captured that p_1 and p_2 are close together.

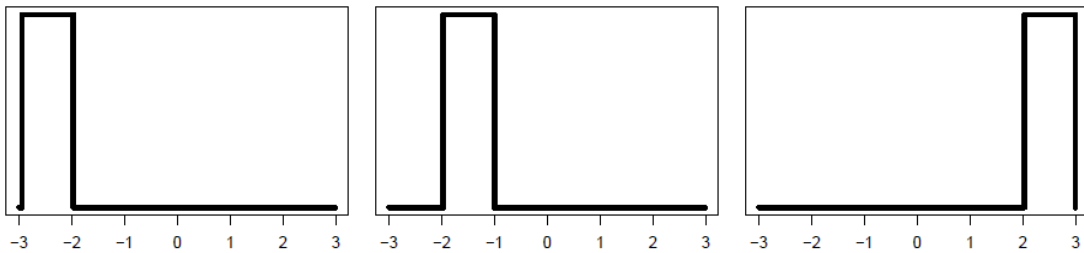


Figure 2.1: Three densities p_1, p_2, p_3 [6].

2.1 Kantorovich Optimal Transport Problem (OTP)

Define a and b are the weight at the locations x_1, \dots, x_n in image X and y_1, \dots, y_m in image Y . The Monge problem [9] seeks a map that associates to each point x_i a single point y_i and which must push the weight of a toward the weight b .

Let $P \in \mathbb{R}_+^{n \times m}$ be a coupling matrix, where $P_{i,j}$ describes the amount of mass flowing from pixel i toward pixel j , or from the weight found at x_i toward y_j . For better understanding, a coupling matrix can be described as a transportation service [13]. In resource distribution scenarios, an operator is tasked with moving goods from a warehouse indexed by i to a factory indexed by j . This service imposes a uniform cost C_{ij} per unit for transferring resources. Possessing exclusive transport rights, the company applies a consistent pricing mechanism across the economic landscape, making the transportation cost for a units amount to $a \times C_{ij}$. In response to these conditions, the operator resolves to apply a linear programming approach as delineated in Equation 2.3 to derive an optimal transportation matrix P^* . This matrix details the quantity P_{ij} of resources to be shipped from warehouse i to factory j .

Admissible couplings admit simple characterization,

$$U(a, b) \stackrel{\text{def}}{=} \{P \in \mathbb{R}_+^{n \times m} : P\mathbf{1}_m = a \text{ and } P^T\mathbf{1}_n = b\}, \quad (2.1)$$

where the following matrix-vector notation is used:

$$P\mathbf{1}_m = \left(\sum_j P_{i,j} \right) \in \mathbb{R}^n \text{ and } P^T\mathbf{1}_n = \left(\sum_i P_{i,j} \right) \in \mathbb{R}^m. \quad (2.2)$$

The set of matrices $U(a, b)$ is bounded and defined by $n + m$ equality constraints, and therefore is a convex polytope (the convex hull of a finite set of matrices) [2].

Kantorovich's relaxed formulation is always symmetric, in the sense that a coupling P is in $U(a, b)$ if and only if P^T is in $U(b, a)$ 2.1. Kantorovich's formulation of optimal transport between 2 measures is the problem between their associated probability weight vectors a, b and the cost matrix C [13]

$$L_C(a, b) \stackrel{\text{def}}{=} \min_{P \in U(a, b)} \langle C, P \rangle \stackrel{\text{def}}{=} \sum_{i,j} C_{i,j} P_{i,j}, \quad (2.3)$$

where the cost matrix is [13]

$$C_{i,j} \stackrel{\text{def}}{=} c(x_i, y_j). \quad (2.4)$$

Solving the Kantorovich's OTP is challenging. Therefore, an entropic regularization needs to be added to approximate Wasserstein distance between 2 measures.

2.2 Entropic Regularization of Optimal Transport

G. Peyre and M. Cuturi in [13] described using a regularizing function to obtain approximate solutions to the original transport problem. This form of regularization presents a multitude of significant benefits that collectively establish it as an exceedingly effective instrument. The optimization of the regularized objective can be conducted through an uncomplicated scheme of alternating minimization. This procedure corresponds to iterations that involve straightforward matrix-vector operations, which are notably well-aligned with the computational architecture of GPUs. For certain scenarios, there is no necessity to retain a full $n \times m$ cost matrix in memory; instead, it suffices to perform evaluations on a kernel basis. Moreover, when a substantial subset of measures possesses identical support, the aforementioned matrix-vector operations can be efficiently transformed into matrix-matrix operations, which yields considerable computational accelerations. The entropic regularization is added to Kantorovich's OTP to approximate Wasserstein's distance between 2 measures.

The discrete entropy of a coupling matrix is defined as

$$H(P) \stackrel{\text{def}}{=} - \sum_{i,j} P_{i,j} (\log(P_{i,j}) - 1), \quad (2.5)$$

with an analogous definition for vectors, with the convention that $H(a) = -\infty$ if one of the entries a_i is 0 or negative. To avoid the infinity in the lower bound an offset is added to the coupling matrix P . The function H is 1-strongly concave, because its Hessian is $\partial^2 H(P) = -\text{diag}(1/P_{i,j})$ and $P_{i,j} \leq 1$. By using $-H$ as a regularizing function, we are able to approximate solutions to the original problem (2.3):

$$L_C^\varepsilon(a, b) \stackrel{\text{def}}{=} \min_{P \in U(a,b)} \langle P, C \rangle - \varepsilon H(P). \quad (2.6)$$

Since the objective is an ε -strongly convex function, Problem (2.6) has a unique optimal solution.

The unique solution P_ε of (2.6) converges to the optimal solution with maximal entropy within the set of all optimal solutions of the Kantorovich problem

$$P_\varepsilon = \underset{P}{\operatorname{argmin}} \{ -H(P) : P \in U(a, b), \langle P, C \rangle = L_C(a, b) \}, \quad (2.7)$$

so that in particular

$$L_C^\varepsilon(a, b) \xrightarrow{\varepsilon \rightarrow 0} L_C(a, b). \quad (2.8)$$

One also has

$$P_\varepsilon \xrightarrow{\varepsilon \rightarrow \infty} a \otimes b = ab^T = (a_i b_j)_{i,j}. \quad (2.9)$$

2.3 Sinkhorn's Algorithm

The following statement shows that the solution of equation (2.6) follows a unique structure that may be represented in terms of $n + m$ variables. Since nm variables within $U(a, b)$ are bounded by $n + m$ constraints on a coupling matrix P , this representation is effectively the dual aspect.

The unique solution P_ε of (2.6) is a projection onto $U(a, b)$ of the Gibbs kernel associated to the cost matrix C as

$$K_{i,j} \stackrel{\text{def}}{=} e^{-\frac{C_{i,j}}{\varepsilon}} \quad (2.10)$$

has the form

$$\forall (i, j) \in [n] \times [m], \quad P_{i,j} = u_i K_{i,j} v_j, \quad (2.11)$$

for two (unknown) scaling variable $(u, v) \in \mathbb{R}_+^n \times \mathbb{R}_+^m$.

It is possible to efficiently translate the factorization of the optimal solution, provided by equation (2.6), into matrix notation where $P = \text{diag}(u)K\text{diag}(v)$. The two variables (u, v) have to satisfy the particular nonlinear equations that correspond with the mass conservation constraints incorporated in $U(a, b)$:

$$\text{diag}(u)K\text{diag}(v)\mathbf{1}_m = a, \quad \text{and} \quad \text{diag}(v)K^T\text{diag}(u)\mathbf{1}_n = b. \quad (2.12)$$

The two given equations can be further simplified by taking the product of $\text{diag}(u)$ with Kv to be an element-wise multiplication since the term $\text{diag}(v)\mathbf{1}_m$ is equal to v . The formulas thus have the form

$$u \odot (Kv) = a \quad \text{and} \quad v \odot (K^T u) = b, \quad (2.13)$$

where \odot stands for the Hadamard product. In numerical analysis, this set of equations is known as the matrix scaling problem (see [10] and references therein). An iterative approach can be used to solve these equations by first modifying u to match the left side of Equation (2.13), and then v to meet the right side. The Sinkhorn's algorithm is made up of these iterative phases and is represented as follows:

$$u^{(\ell+1)} = \frac{a}{Kv^{(\ell)}} \quad \text{and} \quad v^{(\ell+1)} = \frac{b}{K^T u^{(\ell+1)}}, \quad (2.14)$$

beginning with an arbitrary positive vector $v^{(0)} = 1_m$. The division indicated is to be interpreted entrywise. These iterations converge and all result in the same optimal coupling $\text{diag}(u)K\text{diag}(v)$ regardless of different initialization.

To be able to differentiate the algorithm which computes Wasserstein distances:

1. Initialize the vector $V^{(0)}$ to a vector of ones (e).
2. For each iteration l in a loop from 1 to 50, perform the following updates:
 - Compute $s^{(l)}$ by multiplying the matrix K with the vector $v^{(l-1)}$.
 - Update $u^{(l)}$ by element-wise division of the vector a by $s^{(l)}$.
 - Compute $t^{(l)}$ by multiplying the transpose of K with the vector $u^{(l)}$.
 - Update $v^{(l)}$ by element-wise division of the vector b by $t^{(l)}$.
3. After the iterative updates, compute the coupling matrix P using the vectors $u^{(50)}$ and $v^{(50)}$.
4. Compute the entropy term h as the sum of certain function evaluations on the elements of P .
5. Compute the objective function L as a combination of the entropy term and the weighted sum of the cost matrix C using the coupling matrix P .
6. The final objective function f is set to L .

We chose the fixed iteration limit of 50 based on some other experiments not reported here.

In summary, the approach estimates the Wasserstein distance between two measures denoted by the vectors a and b via entropic regularization. The Wasserstein distance is calculated as part of the total objective function, and the regularization term adds entropy to aid in optimization. The methodology known as Sinkhorn's algorithm utilizes an iterative method to determine the coupling matrix. It takes into account restrictions related to mass conservation and incorporates entropic regularization to enhance computing efficiency and convergence.

The Wasserstein distance algorithm is shown below.

Algorithm 1 Regularized Wasserstein metric algorithm

 $V^{(0)} \leftarrow e$ **for** $l = 1 : 50$ **do** $s^{(l)} = K v^{(l-1)}$ $u^{(l)} = \frac{a}{s^{(l)}}$ $t^{(l)} = K^T u^{(l)}$ $v^{(l)} = \frac{b}{t^{(l)}}$ **end for****for** $i = 1 : n$ **do****for** $j = 1 : n$ **do** $P_{ij} = u_i^{(50)} K_{ij} v_j^{(50)}$ **end for****end for** $h = \sum_{i,j=1}^n -\varphi(P_{ij})$ $\triangleright \varphi(P_{ij}) = P_{ij}(\log(P_{ij}) - 1)$ $L = -\varepsilon h + \sum_{i,j=1}^n P_{ij} C_{ij}$ $f = L$

Chapter 3

Autoencoder

The primary principle behind autoencoders [1] is to utilize the input as the objective for reconstruction, thus reproducing the input data as its output. This idea, which was first suggested by Sanjaya Addanki and developed by the PDP group [12, 8], might not appear very important at first because it only entails reproducing an already-existing input. But rather than the output itself, the true value lies in the abstract representations generated in the network's hidden layers. While feedforward autoencoders with a single hidden layer are the main focus of this thesis, autoencoder setups that include recurrent connections and multiple hidden layers are also viable. It is claimed that understanding the linear examples of the single-hidden layer framework is necessary to understand more complex, multi-layered situations. To keep things simple and expand on the analysis of network architectures, a single hidden layer architecture is considered.

3.1 Structure of Autoencoder

The material in this section is based on the book [1]. For any $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the autoencoder with architecture $A(n, m, n)$ transforms an input vector $x \in \mathbb{F}^n$ into an output vector $A \circ B(x) \in \mathbb{F}^n$ (fig. 3.1). The corresponding *autoencoder problem* is to find $A \in \mathcal{A}$ and $B \in \mathcal{B}$ that minimize the overall error or distortion function:

$$\min_{A,B} \mathcal{E}(A, B) = \min_{A,B} \sum_{k=1}^K \mathcal{E}(x_k) = \min_{A,B} \sum_{k=1}^K \Delta(A \circ B(x_k), x_k) \quad (3.1)$$

where:

- n, m and K are positive integers.
- \mathbb{F} and G are sets.
- \mathcal{A} is a class of functions from G^m to \mathbb{F}^n .
- \mathcal{B} is a class of functions from \mathbb{F}^n to G^m .
- $\mathcal{X} = \{x_1, \dots, x_K\}$ is a set of K training vectors in \mathbb{F}^n . External targets presented (hetero-association) by the same set $\mathcal{X} = \{x_1, \dots, x_K\}$ and denoted the corresponding set of target vectors in \mathbb{F}^n . For convenience, the set $\mathcal{Y} = \{y_1, \dots, y_K\}$ will be used as target set.
- Δ is a distance or distortion function (e.g., L_p norm, Hamming distance) defined over \mathbb{F}^n .

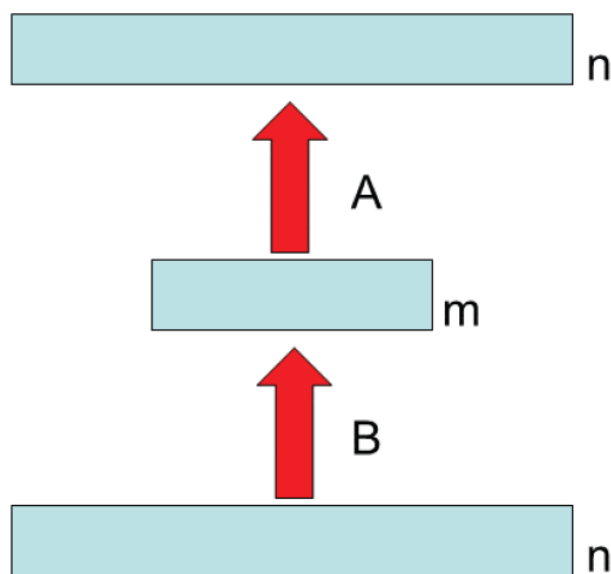


Figure 3.1: An $A(n; m; n)$ Autoencoder Architecture [1].

3.2 Autoencoder vs SVD

Here we consider a compressive linear Autoencoder $A(784, 2, 784)$ over the real numbers, which takes as input a reshaped image of a digit of the size 28×28 . The goal of the

Autoencoder is to compress 1500 images of three different digits of MNIST handwritten digits dataset [7]. In the linear case, the functions \mathcal{A} and \mathcal{B} are represented by matrices with weights over the set of images. The goal is the minimization of the squared Euclidean distance [1]:

$$\min_{A,B} \mathcal{E}(A, B) = \min_{A,B} \sum_{k=1}^K \|x_k - ABx_k\|^2 = \sum_{k=1}^K (x_k - ABx_k)^t (x_k - ABx_k). \quad (3.2)$$

Let Y be the 784×784 input-to-output matrix computed by the linear Autoencoder. As there is a bottleneck layer of two neurons, we must have $\text{rank}(Y) \leq 2$, and probably $\text{rank}(Y) = 2$. We expect that $\|A - YA\|$ is also small since the Autoencoder seems to be performing well. However, since W is optimal, we know that in the Frobenius norm, we must have $\|A - YA\| \geq \|A - XA\|$.

The Frobenius norm shows that the trained linear Autoencoder computes an affine linear transformation to compress the data. This means that there is a matrix $A \in \mathbb{R}^{784 \times 784}$ and a vector $\mathbf{c} \in \mathbb{R}^{784}$ such that if \mathbf{x} is an input to the trained Autoencoder, that is, $\mathbf{x} \in \mathbb{R}^{784}$, then the output to the Autoencoder is of the form $A\mathbf{x} + \mathbf{c}$. Furthermore, because the hidden layer has only two nodes, $\text{rank}(A) \leq 2$.

Given the trained Autoencoder, explicit forms of A and \mathbf{c} were obtained as follows. One runs the trained Autoencoder on the vector of all 0 s, denoted $\mathbf{0} \in \mathbb{R}^{784}$. Then the output should be $A \cdot \mathbf{0} + \mathbf{c} = \mathbf{c}$.

To obtain the compression matrix A , one runs the trained Autoencoder 784 times on the standard basis vectors of \mathbb{R}^{784} , which are

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Let these 784 vectors be denoted $\mathbf{e}_1, \dots, \mathbf{e}_{784}$. Let the 784 vectors output by the trained encoder on $\mathbf{e}_1, \dots, \mathbf{e}_{784}$ be denoted $\mathbf{v}_1, \dots, \mathbf{v}_{784}$. Observe that $\mathbf{v}_i = A\mathbf{e}_i + \mathbf{c} = A(:, i) + \mathbf{c}$. Therefore, we recover $A(:, i)$ as $\mathbf{v}_i - \mathbf{c}$. In other words, the matrix A is the concatenation $A = [\mathbf{v}_1 - \mathbf{c}, \dots, \mathbf{v}_{784} - \mathbf{c}]$.

By determining A and \mathbf{c} , by the procedure above, confirmation of the correctness of computed A and \mathbf{c} are obtained using the following test. Let $\mathbf{x} \in \mathbb{R}^{784}$ be a randomly chosen vector. Let \mathbf{v} be the output from the trained Autoencoder when applied to \mathbf{x} . Let

$\mathbf{v}' := A\mathbf{x} + \mathbf{c}$, where A and \mathbf{c} have been obtained by the above procedure. Then \mathbf{v} and \mathbf{v}' show be very close, in other words, $\|\mathbf{v} - \mathbf{v}'\|/\|\mathbf{v}\|$ should be a very small number.

The SVD computes the optimal linear fitting for the data, but we need an extra step to compute the optimal affine linear fitting. The following procedure to compute an adequate affine linear fitting is a heuristic. Let the data matrix be denoted by $D \in \mathbb{R}^{784 \times 1500}$. In other words, each column of D corresponds to one handwritten digit. The algorithm to compute a rank-2 affine fitting to D :

1. Compute $\mathbf{p} \in \mathbb{R}^{784}$, the average of all the columns of D . If $\mathbf{e} \in \mathbb{R}^{1500}$ denotes the vector of all 1's of length 1500, then this computation may be written as $\mathbf{p} := D\mathbf{e}/1500$.
2. Let \bar{D} be the matrix that results from subtracting \mathbf{p} from each column of D . In matrix notation: $\bar{D} := D - \mathbf{p}\mathbf{e}^T$.
3. Factorize \bar{D} as $\bar{D} = U\Sigma V^T$ (SVD). Here, $U \in \mathbb{R}^{784 \times 784}$ is orthogonal, $\Sigma \in \mathbb{R}^{784 \times 1500}$ is diagonal, and $V \in \mathbb{R}^{1500 \times 1500}$ is orthogonal.
4. The rank-2 affine approximation to D is given by:

$$D^{\text{appx}} = U(:, 1 : 2)\Sigma(1 : 2, 1 : 2)V(:, 1 : 2)^T + \mathbf{p}\mathbf{e}^T.$$

5. The rank-2 compressing matrix for a vector \mathbf{x} determined by the SVD is

$$A' = U(:, 1 : 2)U(:, 1 : 2)^T,$$

which is 784×784 . The full transformation is given by $\mathbf{x} \mapsto A'(\mathbf{x} - \mathbf{p}) + \mathbf{p}$. In other words, the mapping is given by $\mathbf{x} \mapsto A'\mathbf{x} + \mathbf{c}'$ where \mathbf{c}' has the formula $\mathbf{c}' = \mathbf{p} - A'\mathbf{p}$.

Classification of the digits by the linear approximation and affine approximation are shown in Figure 3.2 and Figure 3.2 respectfully.

For optimization, SGD [4] and Adam [5] optimizers were applied to Autoencoder network to perform better compression of handwritten digits dataset [7]. Figure 3.4 shows classifications of the digits by Autoencoder with SGD optimizer in comparison with Autoencoder with Adam optimizer.

Compressed images are shown in table 3.1.

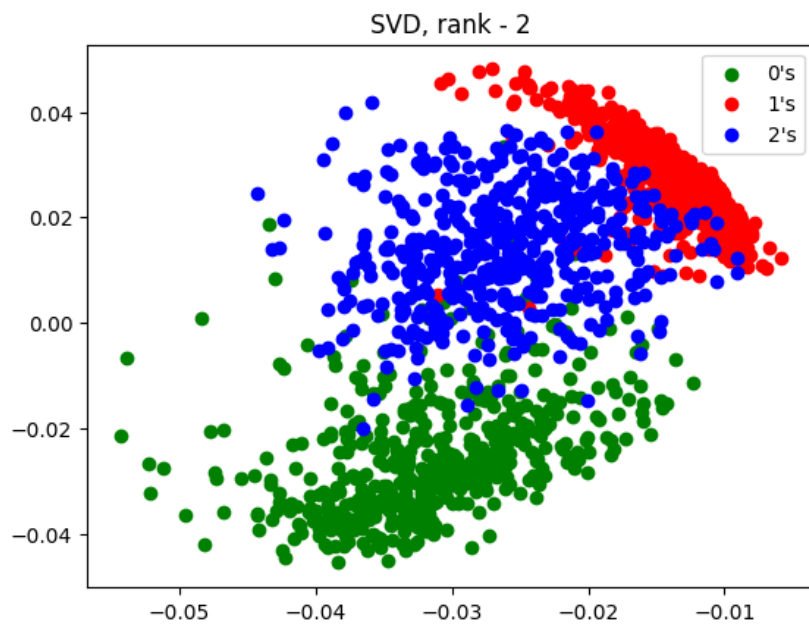


Figure 3.2: Linear SVD with rank 2.

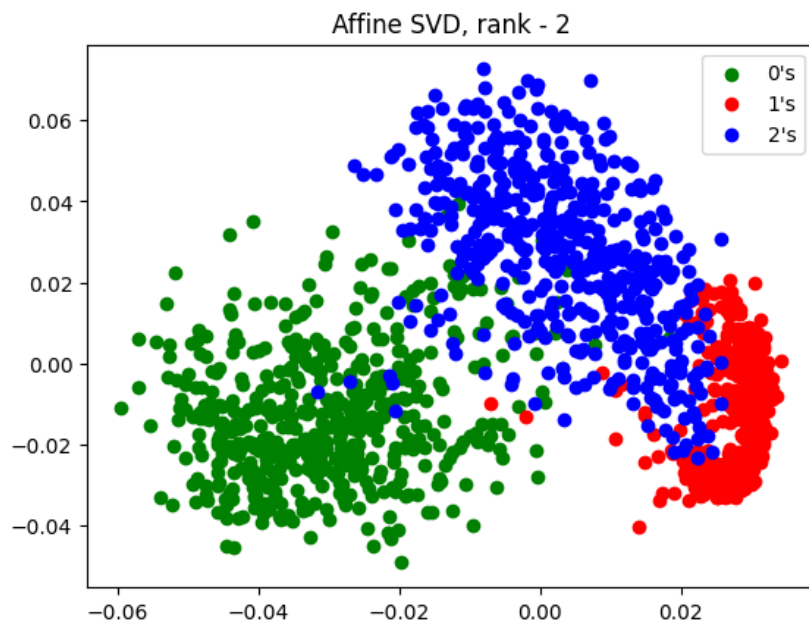


Figure 3.3: Affine SVD with rank 2.

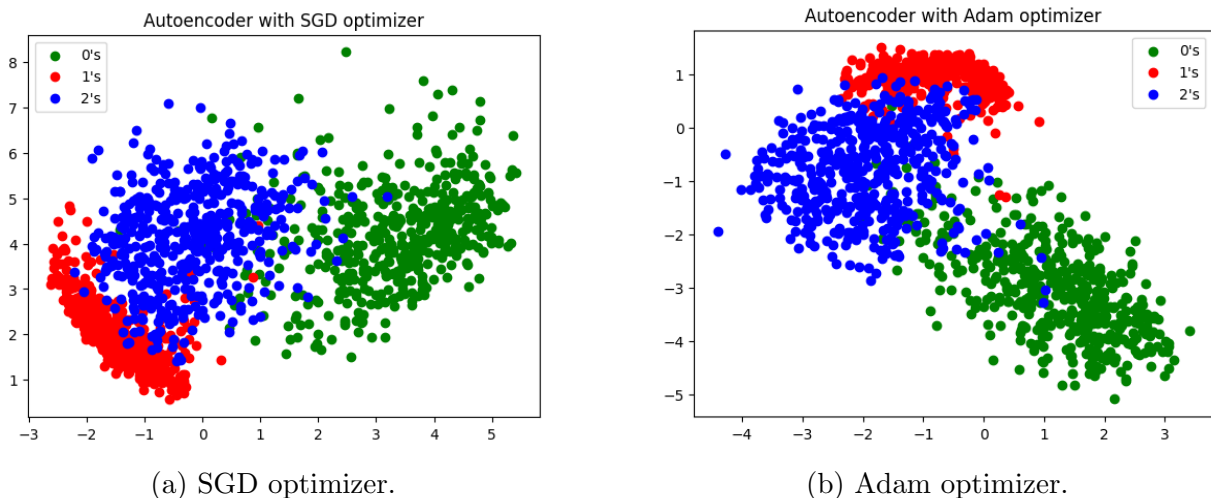


Figure 3.4: Autoencoder with SGD and Adam optimizers.

3.3 Loss function: Wasserstein distance

Used autoencoders with a described previously structure do not substantially outperform mathematical techniques like SVD and Affine SVD, as shown in Table 3.1. In order to do this, the main goal of this work is to investigate the usage of Wasserstein distance in determining the separation between input and output images. Since we are compressing images, the distance between an original image and its compressed form should approach zero, and the image must remain unchanged. The Wasserstein distance is to be used as an update step in SGD [4] and Adam [5] optimization algorithms.

Both SGD and Adam algorithms require a derivative of the Wasserstein distance with respect to the weights of the nodes of Autoencoder. The derivative of the Wasserstein distance f between the input and output images with respect to the weights w of Autoencoder's nodes must be calculated in order to employ Wasserstein distance as a loss function. A derivative of the Wasserstein distance f with respect to an input image a and a derivative of the image a with respect to the weights w of the nodes must be found in order to determine the derivative of the Wasserstein distance f with respect to weights w :

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial w}. \quad (3.3)$$

The first term of the equation 3.3 is the derivative $\frac{\partial a}{\partial w}$ of an image a with respect to the weights w of the nodes, and is a standard back-propagation in order to train Autoencoder.

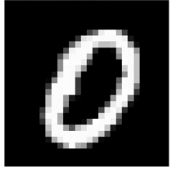
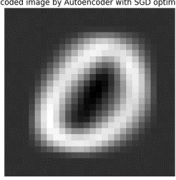
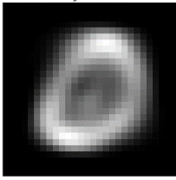
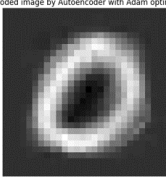
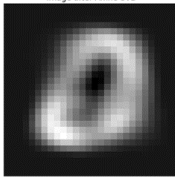
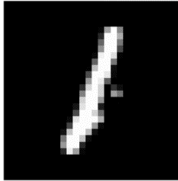
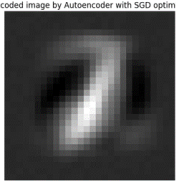
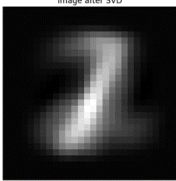
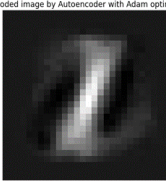
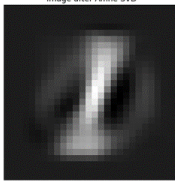
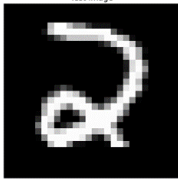
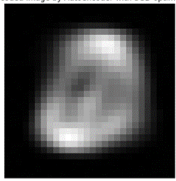
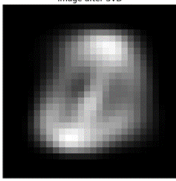
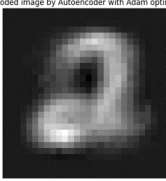
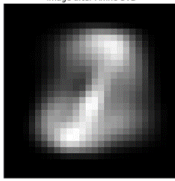
Digit	SGD	SVD	Adam	Affine SVD
<small>Test image</small> 	<small>Decoded image by Autoencoder with SGD optimizer</small> 	<small>Image after SVD</small> 	<small>Decoded image by Autoencoder with Adam optimizer</small> 	<small>Image after Affine SVD</small> 
<small>Test image</small> 	<small>Decoded image by Autoencoder with SGD optimizer</small> 	<small>Image after SVD</small> 	<small>Decoded image by Autoencoder with Adam optimizer</small> 	<small>Image after Affine SVD</small> 
<small>Test image</small> 	<small>Decoded image by Autoencoder with SGD optimizer</small> 	<small>Image after SVD</small> 	<small>Decoded image by Autoencoder with Adam optimizer</small> 	<small>Image after Affine SVD</small> 

Table 3.1: Table of Figures

To compute the second term of the equation 3.3, the derivative $\frac{\partial f}{\partial a}$ of the Wasserstein distance f with respect to an input image a , the autodifferentiation needs to be applied.

3.4 Wasserstein distance: Autodifferentiation

The material in this section is based on the book [11]. The foundation of automatic differentiation techniques is the understanding that evaluating any function is a sequence of basic operations, each of which handles one or two parameters at a time. Addition, multiplication, division, and the power operation a^b are examples of operations that need two inputs. Functions such as logarithmic, exponential, and trigonometric functions are examples of single-argument operations. Another common element across several automated differentiation techniques is the use of the chain rule. The well-known rule from basic calculus asserts that we may express the derivative of h with respect to x as follows:

$$\nabla_x h(y(x)) = \sum_{i=1}^m \frac{\partial h}{\partial y_i} \nabla y_i(x), \quad (3.4)$$

where h is a function of the vector y , which is in turn a function of the vector x [11].

There are two basic modes of automatic differentiation: the forward and reverse modes. The computational complexity of forward-mode automated differentiation would be proportionate to the number of input variables as it would need calculating the derivatives with respect to each input variable separately [11]. In contrast, reverse autodiff is a more effective method for calculating gradients in situations when there are many inputs and only one output, which makes it an appropriate choice for optimization issues such as the Wasserstein metric technique. Reverse autodiff calculates the derivatives of the final output with regard to intermediate variables by traversing from the output back to the input on the computational graph in reverse order.

Generally, the reverse mode of automatic differentiation retrieves the partial derivatives of a function h with respect to all variables (images in the input set) h_i by executing a reverse sweep of the computational graph. The partial derivatives $\frac{\partial h}{\partial h_i}$ with respect to the independent variables $h_i, i = 1, 2, \dots, n$ can be joined to form the gradient vector ∇f at the end of this operation.

An important idea in reverse-mode automated differentiation is the reverse sweep, that relies on the following observation, which is grounded in the chain rule 3.4: The partial derivatives corresponding to any node i 's child nodes j can be used to generate the partial derivative for that node [11]:

$$\frac{\partial h}{\partial x_i} = \sum_{j \text{ a child of } i} \frac{\partial h}{\partial x_j} \frac{\partial x_j}{\partial x_i}. \quad (3.5)$$

As soon as we calculated the right-hand-side term in (3.5) for each node i , we add it to \bar{x}_i ; in other words, we execute the procedure

$$\bar{x}_i \leftarrow \bar{x}_i + \frac{\partial h}{\partial x_j} \frac{\partial x_j}{\partial x_i}. \quad (3.6)$$

The partial derivative of the objective function h with respect to the variable x_i is being updated by this equation (3.6). Contributions are accumulated by multiplying the local sensitivity of x_j with regard to x_i by the partial derivatives of h with respect to its child nodes (x_j). In contrast to the forward calculation of the function h , the computation is carried out in the opposite direction. The partial derivative $\frac{\partial h}{\partial x_i}$ is collected for each node (i) through contributions from its child nodes. When all child nodes have contributed, the node is considered "finalized." The computation of the partial derivative $\frac{\partial h}{\partial x_i}$ occurs when a node (i) is finalized. At that point, the node is prepared to add a term to the

summation for every parent node. The contribution is produced using formula (3.6) for every parent node. Nodes iteratively contribute to their parent nodes until all nodes have been finalized. In the graph, the direction of the computation flow is reversed from the function evaluation flow, going from children to parents.

The main objective of the forward sweep is to compute the numerical values of each variable x_i and evaluate the function h . Additionally, the numerical values of the partial derivatives $\frac{\partial x_j}{\partial x_i}$ are computed and recorded for each arc (connection) in the computational graph. Each variable x_j 's local sensitivities to variations in x_i are represented by these partial derivatives.

Instead of using symbolic formulas or computer code utilizing the variables x_i or the partial derivatives $\frac{\partial h}{\partial x_i}$, the forward sweep works with numerical values. By the time the forward sweep is complete, the system has the numerical values of the partial derivatives $\frac{\partial x_j}{\partial x_i}$ for every arc in the graph in addition to the numerical values of the variables x_i .

These stored partial derivatives have numerical values that are important for the reverse sweep (backpropagation). In order to quickly calculate the gradients $\frac{\partial h}{\partial x_i}$ in the reverse sweep, the previously computed $\frac{\partial x_j}{\partial x_i}$ values acquired during the forward sweep are utilized. These are the numerical numbers used in the equation 3.6 for the reverse sweep. The gradients of the objective function with respect to each variable may be easily computed by the reverse sweep by using the numerical values of the partial derivatives that were saved during the forward sweep. This method improves computing efficiency by avoiding the need to recompute the symbolic derivatives during the reverse sweep.

As a result, the forward sweep computes and saves numerical values for variables and partial derivatives, which are then used to efficiently compute gradients during the reverse sweep.

Algorithm 1 is appropriate for reverse-mode automatic differentiation (reverse autodiff) since it produces the Wasserstein distance as a scalar output through a series of operations. Reverse autodiff involves traversing the computational graph in reverse order, from the output back to the input, in order to calculate the derivatives of the final result with regard to intermediate variables.

In the Wasserstein metric algorithm (1), updating the variables in the loop—specifically, calculating $s^{(l)}$, $u^{(l)}$, $t^{(l)}$, and $v^{(l)}$ —is essential for determining the Wasserstein distance. Matrix-vector products and element-wise operations are used in these computations. These factors are combined to form the Wasserstein distance (L).

In this case, reverse autodiff works well because it makes it possible for the algorithm to compute gradients quickly by going backwards from the final result (L) through the

procedures. This is helpful because it prevents repetitive calculations by reusing intermediate findings during the backward pass, which is particularly useful when there are several input variables (as in the case of Wasserstein metric computations) and a single output.

To differentiate the function which implements Wasserstein algorithm 1, the gradients of the objective function f with respect to its inputs should be computed. The input vectors a and b to the function which implements Algorithm 1 are used as an original image and the compressed image.

To differentiate the algorithm 1:

1. Initialize by setting $\frac{\partial f}{\partial L} = 1$ since L is the final output.
2. Calculate $\frac{\partial f}{\partial h} = -\varepsilon \frac{\partial f}{\partial L}$.
3. Update $\frac{\partial f}{\partial P_{ij}} = C_{ij} \frac{\partial f}{\partial L}$ for all i, j .
4. Adjust $\frac{\partial f}{\partial P_{ij}}$ using the chain rule: $\frac{\partial f}{\partial P_{ij}} = \frac{\partial f}{\partial P_{ij}} - \frac{\partial f}{\partial h} \varphi' (P_{ij})$ where $\varphi' (P_{ij})$ is the derivative of the function φ defined in the algorithm.
5. Compute $M = \frac{\partial f}{\partial P} \cdot K$, where K is a matrix.
6. Update $\frac{\partial f}{\partial u^{(50)}} = M v^{(50)}$ and $\frac{\partial f}{\partial v^{(50)}} = M^T u^{(50)}$.
7. Iterate backward through the loop from $l = 50$ to $l = 1$:
 - Compute $t^{(l)} = \frac{\partial f}{\partial v^{(l)}} \cdot \left(-\frac{b}{(t^{(l)})^2} \right)$.
 - For $l = 50$, update $\frac{\partial f}{\partial u^{(l)}} += K \frac{\partial f}{\partial t^{(l)}}$, otherwise set $\frac{\partial f}{\partial u^{(l)}} = K \frac{\partial f}{\partial t^{(l)}}$.
 - Update $\frac{\partial f}{\partial s^{(l)}} = \frac{\partial f}{\partial u^{(l)}} \cdot \left(-\frac{a}{(s^{(l)})^2} \right)$.
 - Update $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial a} + \frac{\partial f}{\partial u^{(l)}} \cdot \left(\frac{1}{s^{(l)}} \right)$.
 - Update $\frac{\partial f}{\partial v^{(l-1)}} = K^T \frac{\partial f}{\partial s^{(l)}}$.

This process essentially computes the gradients of the objective function f with respect to the inputs \mathbf{a} and \mathbf{b} by traversing the computational graph in reverse order. The algorithm that implements the steps described above is presented as Algorithm 2.

The truncation error is used to verify the accuracy of the autodifferentiation process and the sensitivity of the Wasserstein distance function with perturbations in the input

Algorithm 2 Reverse Mode Autodiff of Wasserstein metric algorithm

$$\frac{\partial f}{\partial L} = 1$$

$$\frac{\partial f}{\partial h} = -\varepsilon \frac{\partial f}{\partial L}$$

$$\frac{\partial f}{\partial P_{ij}} = C_{ij} \frac{\partial f}{\partial L} \quad \triangleright \forall i, j$$

$$\frac{\partial f}{\partial P_{ij}} = \frac{\partial f}{\partial P_{ij}} - \frac{\partial f}{\partial h} \varphi'(P_{ij}) \quad \triangleright \varphi'(P_{ij}) = \begin{cases} 0, & \text{if } x = 0 \\ \log(P_{ij}), & \text{otherwise} \end{cases}$$

$$M = \frac{\partial f}{\partial P} \cdot K$$

$$\frac{\partial f}{\partial u^{(50)}} = M v^{(50)}$$

$$\frac{\partial f}{\partial v^{(50)}} = M^\top u^{(50)}$$
for $l = 50 : -1 : 1$ **do**

$$t^{(l)} = \frac{\partial f}{\partial v^{(l)}} \cdot \left(-\frac{b}{(t^{(l)})^2} \right)$$
if $l = 50$ **then**

$$\frac{\partial f}{\partial u^{(l)}} + = K \frac{\partial f}{\partial t^{(l)}}$$
else

$$\frac{\partial f}{\partial u^{(l)}} = K \frac{\partial f}{\partial t^{(l)}}$$
end if

$$\frac{\partial f}{\partial s^{(l)}} = \frac{\partial f}{\partial u^{(l)}} \cdot \left(-\frac{a}{(s^{(l)})^2} \right)$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial a} + \frac{\partial f}{\partial u^{(l)}} \cdot \left(\frac{1}{s^{(l)}} \right)$$

$$\frac{\partial f}{\partial v^{(l-1)}} = K^T \frac{\partial f}{\partial s^{(l)}}$$
end for

image a . A set of regularization parameters λ is chosen: $0.1, 0.01, \dots, 10^{-10}$. For each λ , the Wasserstein distance function is evaluated with a perturbed input $a + \lambda\Delta a$, where $\Delta a \in \mathbb{R}^n$ is a random normalized vector.

$$\frac{f(a + \lambda\Delta a) - f(a)}{\lambda} - \left\langle \frac{\partial L}{\partial a}, \Delta a \right\rangle, \tag{3.7}$$

where $\Delta L \approx \frac{\partial L}{\partial a} \cdot \Delta a$.

The test aims for the error to approach zero as λ approaches zero.

The results of the truncation error are shown in Table 3.2 and are indicating convergence of the numerical approximation to the true derivative. However, for extremely small values of λ , there is a divergence, and the truncation error becomes negative. This could be an indication of numerical instability or limitations in the precision of the calculations.

λ	Trunc error
1	$9.149 * 10^{-2}$
10^{-1}	$1.508 * 10^{-1}$
10^{-2}	$9.886 * 10^{-2}$
10^{-3}	$1.478 * 10^{-2}$
10^{-4}	$1.547 * 10^{-3}$
10^{-5}	$1.554 * 10^{-4}$
10^{-6}	$1.570 * 10^{-5}$
10^{-7}	$3.220 * 10^{-6}$
10^{-8}	$-5.662 * 10^{-6}$
10^{-9}	$-5.007 * 10^{-5}$
10^{-10}	$-9.169 * 10^{-4}$

Table 3.2: Truncation Error of the Algorithm 2

The next step in the project would be to connect the Wasserstein loss to an autoencoder for the purpose of digit classification. Due to time limitations, we did not implement that step. In the next section, we show some preliminary results about classification using multidimensional scaling.

Chapter 4

Observations

The Wasserstein metric f is computed using the Algorithm 1, which determines the distance between two probability distributions, in this case, two images a and b . In order to minimize the cost L , the method iterates to find a scaling solution u and v to optimize the transport plan P (coupling matrix).

Table 4.1 presents the results of experiments comparing results of applying Wasserstein distance Algorithm 1 for different digit pairs, "0 against 2", "0 against 0", "1 against 2". The experiments vary in terms of different digit images, offset values, and epsilon values.

Experiment	# of iterations	Offset	Epsilon	Wasserstein	Norm
0 against 2	50	0.05	0.05	0.995327	0.052788
0 against 0	50	0.05	0.05	0.173436	0.000000
1 against 2	50	0.05	0.05	1.440770	0.079627
0 against 2	50	0.05	0.10	0.755230	0.052788
0 against 0	50	0.05	0.10	-0.110661	0.000000
1 against 2	50	0.05	0.10	1.492173	0.079627
0 against 2	50	0.10	0.05	0.653446	0.043741
0 against 0	50	0.10	0.05	0.025347	0.000000
1 against 2	50	0.10	0.05	0.825891	0.058365
0 against 2	50	0.10	0.10	0.399932	0.043741
0 against 0	50	0.10	0.10	-0.262807	0.000000
1 against 2	50	0.10	0.10	0.813571	0.058365

Table 4.1: Algorithm 1 experiment with various parameter values.

Notably, experiments with $\text{offset} = 0.05$ and $\text{epsilon} = 0.1$, as well as $\text{offset} = 0.1$ and $\text{epsilon} = 0.05$, yield the most promising results. These two configurations show better distinguishing ability between less similar digits (0 and 2) and smaller distance values for more similar digits (0 against 0). Specifically, for "0 against 2" and "1 against 2," these configurations exhibit values that suggest better discrimination between dissimilar digits and more subtle distinctions among similar digits.

The offset and epsilon values of 0.05 and 0.1, respectively, seem to contribute to the Wasserstein distances' effectiveness in distinguishing between digits in comparison with 2-Norm. These configurations appear to strike a balance, resulting in more meaningful and interpretable Wasserstein distance values for digit comparisons. These insights could guide the selection of parameters for Wasserstein distance calculations, emphasizing the importance of considering offset and epsilon values to achieve optimal discrimination between digit pairs.

To analyze the data and visualize the distances, Multidimensional Scaling (MDS) must be used.

In MDS, normalization, symmetrization, and distance matrix calculation are essential preparatory procedures. It is possible that the origin is meaningless and that the dissimilarity values in the input matrix f are on different scales. By centering the dissimilarities around zero by normalization, the MDS solution becomes invariant to additive constants. Normalization guarantees that the MDS analysis does not concentrate on the absolute magnitudes of the dissimilarities, but rather on their relative differences.

There is a chance that the dissimilarity between a and b will not be the same as the one between b and a . The process of symmetrization guarantees the symmetry of the final MDS solution. In addition to making analysis easier, symmetrization ensures that for both a and b , the coordinates in the reduced-dimensional space remain consistent.

It is vital to do normalization and symmetrization to guarantee that the input dissimilarity matrix is suitably ready for MDS analysis. By doing these stages, the MDS solution becomes more stable and interpretable, which improves its capacity to capture the underlying geometric structure of the data.

In classical MDS [16] uses the knowledge that the coordinate matrix X may be obtained from $B = XX'$ by eigenvalue decomposition. Additionally, double centering may be used to calculate the matrix B from the proximity matrix D .

Steps of a Classical MDS algorithm [17]:

1. Set up the squared proximity matrix $D^{(2)} = [d_{ij}^2]$. Building the double-centered

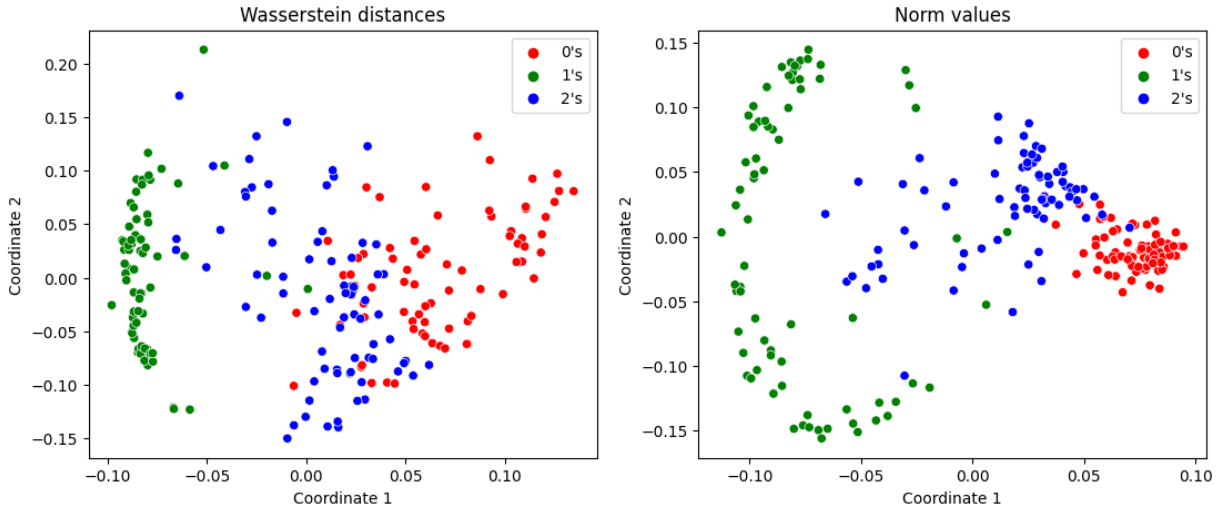


Figure 4.1: Multidimensional scaling of Wasserstein distance and 2-norm

matrix required for the eigenvalue decomposition starts with the squared dissimilarity matrix $D^{(2)}$.

2. Double centering is used as follows: $B = -\frac{1}{2}CD^{(2)}C$ using $C = I - \frac{1}{n}$ as the centering matrix. J_n , where I is the $n \times n$ identity matrix, n is the number of objects, and J_n is a $n \times n$ matrix containing all ones.
3. Find the m biggest eigenvectors e_1, e_2, \dots, e_m and associated eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_m$ of B , where m is the number of required dimensions for the output. Since there are two nodes in the autoencoder's bottleneck and that the rank of the SVD approximation matrix is two, $m = 2$.
4. Now, $X = E_m \Lambda_m^{1/2}$, in which Λ_m is the diagonal matrix of m eigenvalues of B and E_m is the matrix of m eigenvectors.

Classical MDS preserves dissimilarities while reducing the dimensionality of the data and visualizing it in a lower-dimensional environment. The MDS is conducted on both the Wasserstein distances and the normalized distances (fig. 4.1). Based on Wasserstein distances, the first figure displays the data points in a condensed two-dimensional space. In terms of the Wasserstein distance, the clustering or dispersion shows how close or dissimilar the data points are. Following the reduction of dimensionality, the axes indicate the two most important metrics.

The second figure performs the same function for normalized distances. The purpose of these plots is to display how the data distributes or clusters according to the relevant criteria. The associations based on normalized distances are shown in this graphic. It offers an alternative interpretation of the data by emphasizing normalized metrics.

There seems to be a clear point grouping with some overlap across groups in the Wasserstein distance plot. While the '1's' (green points) form a distinct group on the left side of the plot, the '0's' (red points) are more dispersed and overlap with the '2's' (blue points).

The groups are more clearly divided from one another in the 2-norm plot. The '2's' are centred, the '0's' form a tight cluster on the right, and the '1's' are clearly identifiable on the left. When contrasted with the Wasserstein distances, this suggests a more definite distinction across groups.

If the goal is to distinguish between the three categories, the norm values might be considered a more suitable metric, given the clearer separation.

However, the Wasserstein distances could be revealing more subtle relationships between the '0's' and '2's', which might be important for using Wasserstein distance as a loss function in a neural network.

The Wasserstein distances plot's point density indicates a gradient or transition between the '2's' and '0's', which may allude to a spectrum of similarity rather than discrete groupings.

According to the 2-norm plot, each digit appears to have a more distinct identity, with less of a gradient and more of a categorical difference.

It is also worth considering that while the 2-norm values show better separation, it may be overfitting the data if these distinctions are not representative of underlying patterns in larger or more complex datasets.

The Wasserstein distances may be capturing more complex correlations including the structure of the data distribution rather than just distance. Conversely, norm values only consider the geometric separations between the data points.

For example in the annotated Figure 4.2, some '2's' are more similar to '0's' rather than to other '2's' being compared by Wasserstein distance (fig. 4.2a). The annotated graph also shows how '1's' are having separation in Figure 4.2b due to different angles of writing, which should be centred closer as a representation of the same digit.

In summary, the 2-norm appears to be less effective for '1's', excellent for '0's', and moderately effective for '2's'. Wasserstein distance, however, shows a more uniform behaviour,

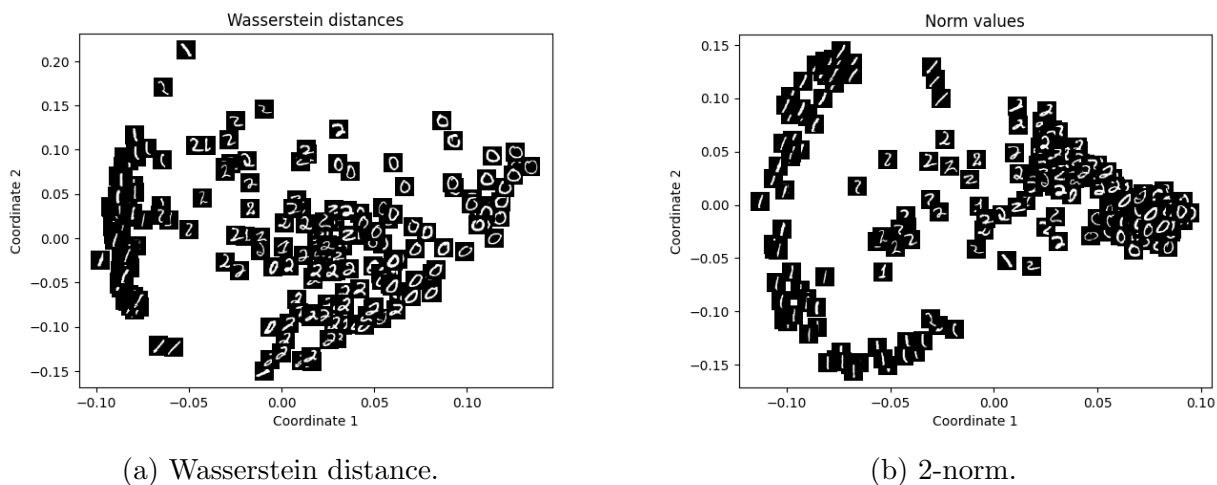


Figure 4.2: Annotated figures of Wasserstein distance and 2-norm.

being moderate for '0's, '1's, and '2's. Wasserstein appears more consistent across categories, making it potentially more suitable for training an autoencoder due to its uniform behaviour across different classes.

The results, which exhibit consistent behaviour across many digit groups, imply that Wasserstein distance may be a more useful method for classifying images. Wasserstein distances provide a more consistent depiction of digit associations, according to the MDS analysis, which may make them a good option for a loss function in an autoencoder. The project's next stages may entail using Wasserstein distances as the loss function in an autoencoder for image classification and image compression, given its promising performance. Improved classification accuracy might result from this, particularly in situations where it's critical to detect minute variations between images. Future studies on Wasserstein distance-based autoencoders may benefit from more investigation and testing.

References

- [1] Pierre Baldi. *Deep learning in science*. Cambridge University Press, 2021.
- [2] Richard A Brualdi. *Combinatorial matrix classes*, volume 13. Cambridge University Press, 2006.
- [3] Gene H. Golub and Van Loan Charles F. *Matrix computations*. Johns Hopkins Univ Press, 2013.
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [6] S. Kolouri. *Optimal Transport and Wasserstein Distance*.
- [7] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [8] Aurora KR LePort, Aaron T Mattfeld, Heather Dickinson-Anson, James H Fallon, Craig EL Stark, Frithjof Kruggel, Larry Cahill, and James L McGaugh. Behavioral and neuroanatomical investigation of highly superior autobiographical memory (hsam). *Neurobiology of learning and memory*, 98(1):78–92, 2012.
- [9] Gaspard Monge. *Memoire sur la theorie des deblais et des remblais*. De l’Imprimerie Royale, 1781.
- [10] Arkadi Nemirovski and Uriel Rothblum. On complexity of matrix scaling. *Linear Algebra and its Applications*, 302:435–460, 1999.
- [11] Jorge Nocedal and Stephen J. Wright. *Calculating Derivatives*, page 193–219. Springer, 2006.

- [12] Osvaldo Olmea and Alfonso Valencia. Improving contact predictions by the combination of correlated mutations and other sources of sequence information. *Folding and design*, 2:S25–S32, 1997.
- [13] Gabriel Peyre and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.
- [14] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.
- [15] Yijing Watkins, Oleksandr Iaroshenko, Mohammad Sayeh, and Garrett Kenyon. Image compression: Sparse coding vs. bottleneck autoencoders. In *2018 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, pages 17–20. IEEE, 2018.
- [16] Florian Wickelmaier. An introduction to mds. *Sound Quality Research Unit, Aalborg University, Denmark*, 46(5):1–26, 2003.
- [17] Wikipedia. Multidimensional scaling — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Multidimensional%20scaling&oldid=1181496234>, 2024. [Online; accessed 25-January-2024].
- [18] Song Zebang and Kamata Sei-Ichiro. Densely connected autoencoders for image compression. In *Proceedings of the 2nd International Conference on Image and Graphics Processing*, pages 78–83, 2019.