

# Neural Approximations to Gromov-Hausdorff Distances

**Bariş Onarici**

Thesis for the attainment of the academic degree

**Master of Science**

at the TUM School of Computation, Information and Technology of the Technical University of Munich

**Supervisor:**

Dr. Steffan Bauer

**Advisors:**

Dr. Bastian Rieck

**Submitted:**

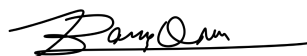
Munich, 31/10/2023



I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 31/10/2023

Bariş Onarıcı

A handwritten signature in black ink, appearing to read 'Bariş Onarıcı', written over a horizontal line.



## Abstract

In this paper, we aim to approximate a lower bound to the Gromov-Hausdorff distances between finite metric spaces, using neural networks. The Gromov-Hausdorff distance is extremely difficult to compute exactly, and even calculating some lower bounds are not computationally feasible. We thus aim to approximate the so-called modified Gromov-Hausdorff distance, introduced by Mémoli, which is a lower bound that is computationally easier to calculate. We do this by utilizing the "Structural Theorem", again introduced by Mémoli, which connects computing the modified Gromov-Hausdorff distance to calculating the Hausdorff distance between the curvature sets of the metric spaces with the  $l^\infty$  metric. This is still computationally infeasible, so we approximate this Hausdorff distance as well, in order to create the dataset our model will learn. We show that only the largest curvature sets should be considered when making this calculation. We utilize heuristic methods such as simulated annealing and genetic algorithms for computing the minimum  $l^\infty$  distance between two curvature sets. We then use a Siamese Neural Network model that takes the distance vectors (upper part of the distance matrices) of the two metric spaces and the modified Gromov-Hausdorff distance between them as input, feeds them into two subnetworks to obtain embeddings, which are concatenated and passed into fully connected layers. The output is an approximation of the modified Gromov-Hausdorff distance between the original metric spaces.

We also use our model to attempt to discriminate between different families of graphs, by using the Floyd-Warshall algorithm to turn adjacency matrices of graphs into shortest path distance matrices. Then, using the model on pairs of distance matrices, we generate histograms depicting the distribution of modified GH distances between the metric spaces obtained from different families of graphs and measuring their dissimilarities.



# Contents

- 1 Introduction** **1**
  - 1.1 Motivation . . . . . 1
  - 1.2 Background . . . . . 2
  - 1.3 Equivalent Formulations and some Bounds of the Gromov-Hausdorff Distance . . . . . 5
  - 1.4 The Modified Gromov-Hausdorff Distance and Curvature Sets . . . . . 8
  
- 2 Estimating the Modified Gromov-Hausdorff Distance** **13**
  - 2.1 The Setup and the Expectation of the Modified Gromov-Hausdorff Distance . . . . . 13
  - 2.2 Creating the Dataset . . . . . 16
  
- 3 The Model** **23**
  - 3.1 Introduction to Neural Networks . . . . . 23
  - 3.2 The Structure of the Model . . . . . 24
  - 3.3 Training the Model . . . . . 25
    - 3.3.1 Different Dimensions . . . . . 26
    - 3.3.2 Dropout and Batch Normalization . . . . . 27
  
- 4 Using the Model to Discriminate between Different Types of Graphs** **29**
  - 4.1 A Brief Introduction to Graphs . . . . . 29
    - 4.1.1 Basic Definitions . . . . . 29
  - 4.2 The Setup . . . . . 32
  
- Bibliography** **39**





# 1 Introduction

## 1.1 Motivation

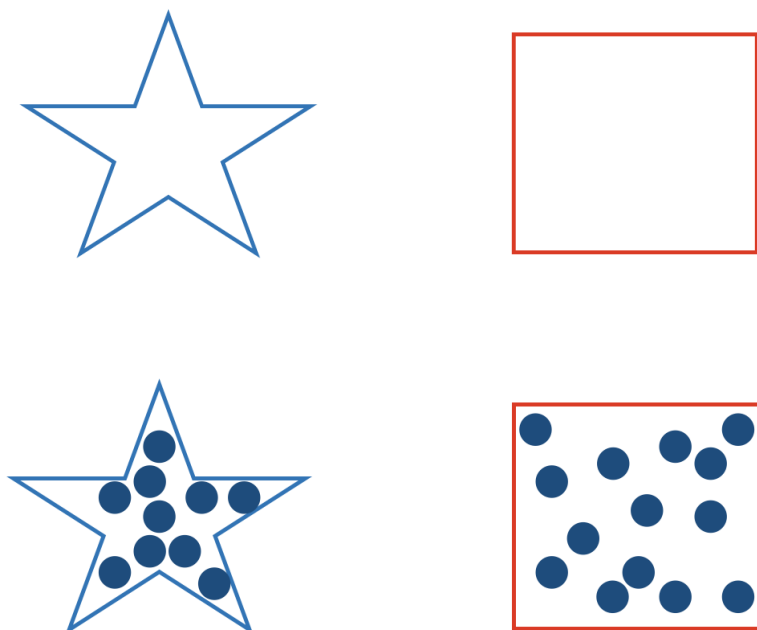
The Gromov-Hausdorff distance measures how far away from being isometric two (compact) metric spaces are, and it is a very useful tool both in pure and applied mathematics. Gromov uses this notion (and the notion of pointed Gromov-Hausdorff convergence [Bel96]) to prove that any discrete group with polynomial growth contains a nilpotent group of finite index. [Gro81]

Moreover, the Gromov-Hausdorff space, which is a metric space of the isometry classes of compact metric spaces with the Gromov-Hausdorff distance, with the topology generated by the Gromov-Hausdorff distance via the Gromov-Hausdorff convergence has some useful properties such as Gromov's compactness theorem [BBI01]. It also has applications in the theory of large deviations. [KS06]

In applied mathematics, the Gromov-Hausdorff distance can be used for *shape matching*, by regarding shapes as compact metric spaces [Mém07] using the necessary metrics for the desired invariances. The GH Distance can then be used on these shapes (as a metric in the space of compact metric spaces) with good theoretical properties. One property of the GH distance is *stability*: [Mém07]

$$|d_{\mathcal{GH}}(X, Y), d_{\mathcal{GH}}(\mathbb{X}_n, \mathbb{Y}_m)| \leq r(\mathbb{X}_n) + r(\mathbb{Y}_m),$$

where  $\mathbb{X}_n \subseteq X$  and  $\mathbb{Y}_m \subseteq Y$  are finite samplings, and  $r(\mathbb{X}_n)$  and  $r(\mathbb{Y}_m)$  are the covering radii.



**Figure 1.1** Example of shape matching: The Gromov-Hausdorff distance provides a dissimilarity metric with good theoretical properties. The second figure shows the shapes with sampled points inside the shapes.

Despite the nice theoretical properties, computing the Gromov-Hausdorff distance is a very difficult task. It is known [Mém07] that the computation of it exactly between finite metric spaces is NP-hard.

Using an alternative expression for the Gromov-Hausdorff distance [KO99], after relaxing and modifying the expression [Mém12], we can get a lower bound to the Gromov-Hausdorff distance between two metric spaces, that is computationally much more feasible. Using the notion of *curvature sets*, it is possible to prove [Mém12] that this lower bound can be computed by comparing the curvature sets of the two metric spaces via their Hausdorff distance.

Our approach will be to discover if neural networks can be used to learn this lower bound to the Gromov-Hausdorff distance between finite metric spaces, and to see if this approach speeds up the computations.

The entire codebase is written in Python, and the neural network implementations are written in the PyTorch library. The graph figures were generated using the NetworkX library in Python and the tikz package in LaTeX.

## 1.2 Background

We work with metric spaces. A *metric space* is a pair  $(X, d)$  where  $X$  is a set and  $d : X \times X \rightarrow \mathbb{R}^+$  is a function satisfying the following properties for all  $x, y, z \in X$ :

- $d(x, y) \geq 0$  and  $d(x, y) = 0$  if and only if  $x = y$ .
- $d(x, y) = d(y, x)$ .
- $d(x, z) \leq d(x, y) + d(y, z)$ .

Here, the function  $d$  is called a metric. A *pseudometric* allows the distance between two distinct points to be zero as well. A pair  $(X, d)$  where  $X$  is a set and  $d : X \times X \rightarrow \mathbb{R}^+$  is a pseudometric is called a pseudometric space.

Let  $s \in X$ , where  $(X, d)$  is a metric space. The *open ball* centered at  $s$  with radius  $r$ , denoted  $B_r(s)$  is the set

$$B_r(s) \triangleq \{x \in X : d(x, s) < r\}$$

Let  $S \subseteq X$ . The  $r$ -neighbourhood of  $S$  in  $X$ , denoted  $U_r(S)$  is the set defined by

$$U_r(S) \triangleq \{x \in X : \text{dist}(x, S) < r\}$$

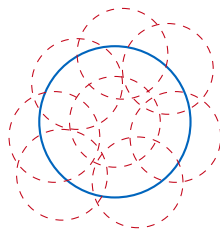
or equivalently,

$$U_r(S) = \bigcup_{x \in S} B_r(x)$$

A *Cauchy sequence* is a sequence  $\{x_n\}$  in a metric space  $(X, d_X)$  such that for all  $\epsilon > 0$  there exists  $n_0 \in \mathbb{N}$  with  $d(x_n, x_m) < \epsilon$  if  $n \geq n_0$  and  $m \geq n_0$ .

$X$  is called *complete* if every Cauchy sequence in  $X$  converges to a point in  $X$ .

An  $\epsilon$ -net of a metric space  $(X, d)$  is any set  $S \subseteq X$  such that for all  $x \in X$  there exists  $s \in S$  with  $d(x, s) \leq \epsilon$ . If there exists a finite  $\epsilon$ -net of  $X$ ,  $X$  is said to be *totally bounded*.



**Figure 1.2** A 2D illustration of compactness, with the more common definition: A set is compact if every *open covering* of that set has a *finite subcovering*. Here, the blue circle represents the entire disk for visibility purposes, and the red dashed circles represent an open covering.

$(X, d)$  is *complete* if every Cauchy sequence of points in  $X$  has a limit that is also in  $X$ .  $X$  is *compact* if it is complete and totally bounded.

If  $(X, d)$  is a compact metric space, then we define its *diameter*, denoted  $\text{diam}(X)$  by  $\text{diam}(X) = \max_{x, x' \in X} d(x, x')$ , its *separation*, denoted  $\text{sep}(X)$  by  $\text{sep}(X) = \inf_{x \neq x'} d(x, x')$ , its *circum-radius*, denoted  $\text{rad}(X)$  by  $\text{rad}(X) = \min_x \max_{x'} d(x, x')$ , and its *eccentricity function*, denoted  $\text{ecc}(X)$  by  $\text{ecc}_X : X \rightarrow \mathbb{R}, x \mapsto \max_{x'} d(x, x')$ .

Given a map  $f : X \rightarrow Y$  between the metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ , its *distortion* is defined as

$$\text{dis}(f) \triangleq \sup_{x, x' \in X} |d_X(x, x') - d_Y(f(x), f(x'))|$$

$f$  is called *Lipschitz* if there exists  $K \geq 0$  such that for all  $x_1, x_2 \in X$ ,

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$$

Any  $K$  that satisfies this is called a Lipschitz constant of  $f$ . The minimal such  $K$  is called the dilation of  $f$  and is denoted  $\text{dil } f$ . Importantly, every Lipschitz map is continuous. [BBI01]

**Proposition.** Let  $(X, d_X)$  and  $(Y, d_Y)$  be metric spaces, where  $Y$  is complete. Let  $S \subseteq X$  be a dense subset of  $X$  and  $f : S \rightarrow Y$  be a Lipschitz map. Then, there exists a unique Lipschitz map  $\tilde{f} : X \rightarrow Y$  with  $\tilde{f}|_S = f$  and  $\text{dil } \tilde{f} = \text{dil } f$ .

*Proof.* Let  $K$  be a Lipschitz constant of  $f$ . For all  $x \in X$ , define  $\tilde{f}(x) \in Y$  by first choosing a sequence  $\{x_n\}_{n=1}^{\infty}$  with  $x_n \in S$  for all  $n$  and  $x_n \rightarrow x$  as  $n \rightarrow \infty$ .

Since  $f$  is Lipschitz,  $d_Y(f(x_i), f(x_j)) \leq K d_X(x_i, x_j)$  for all  $i, j$ . Since  $\{x_n\}$  converges, we have that  $d_X(x_i, x_j) \rightarrow 0$  as  $i, j \rightarrow \infty$  and so  $\{f(x_n)\}$  is a Cauchy sequence in  $Y$ , and hence it converges. Define  $\tilde{f} : X \rightarrow Y$  by  $\tilde{f}(x) = \lim_{n \rightarrow \infty} f(x_n)$ .

Now, for all  $x, x' \in X$ , again letting  $x = \lim x_n$  and  $x' = \lim x'_n$ , we see that

$$d_Y(\tilde{f}(x), \tilde{f}(x')) \lim_{n \rightarrow \infty} d_Y(f(x_n), f(x'_n)) \leq K \lim_{n \rightarrow \infty} d_X(x_n, x'_n) = K d_X(x, x')$$

And so we see that  $\tilde{f}$  is Lipschitz with  $\text{dil } \tilde{f} \leq K$ . Since  $S$  is dense and all such  $\tilde{f}$  will coincide on  $S$ , they will coincide on  $X$  since they are continuous. Thus  $\tilde{f}$  is unique.  $\square$

We will make use of the following notation: (following [Mém12])  $\mathbf{D}_X$  is the map that assigns to each finite subset  $\mathbb{X}$  of a metric space  $(X, d_X)$ , the distance matrix of that subset, so

$$\mathbf{D}_X(\mathbb{X}) = ((d_X(x, x'))_{x, x' \in \mathbb{X}})$$

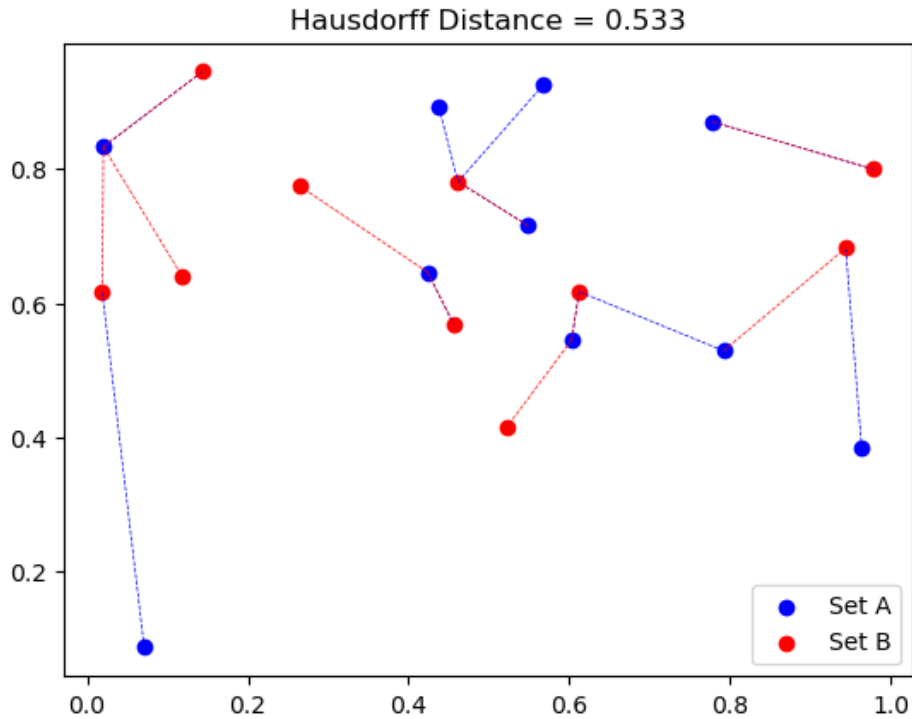
If we were to sample finitely many points from a metric space  $(X, d_X)$  with replacement, then  $\mathbf{D}_X(\mathbb{X})$  would be the distance matrix of a pseudometric space.

We will use the concept of the Hausdorff distance between two subsets of a metric space extensively in our calculations, as well as in defining what the Gromov-Hausdorff distance is. Let  $(X, d)$  be a metric space and let  $A, B \subseteq X$ . The *Hausdorff distance* between  $A$  and  $B$ , denoted  $d_{\mathcal{H}}(A, B)$ , is defined by

$$d_{\mathcal{H}}(A, B) \triangleq \inf\{r > 0 : A \subseteq U_r(B) \text{ and } B \subseteq U_r(A)\}$$

This is equivalent to the following definition

$$d_{\mathcal{H}}(A, B) \triangleq \max \left\{ \sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b) \right\}$$



**Figure 1.3** Depiction of the calculation of Hausdorff distance between two randomly generated sets. The blue dashed line shows the distance from an element of Set A to the closest element of Set B, and similarly the red dashed line shows the distance from an element of Set B to the closest element of set A. The Hausdorff distance is then the maximum of all these distances.

**Proposition** ([BBI01]).  $d_{\mathcal{H}}$  is a metric on the set of compact subsets of a compact metric space  $(X, d)$ .

*Proof.* Clearly,  $d_{\mathcal{H}}$  is nonnegative and symmetric. The triangle inequality follows from the fact that for any  $A \subseteq X$  and  $r_1, r_2 > 0$ ,

$$U_{r_1}(U_{r_2}(A)) \subseteq U_{r_1+r_2}(A)$$

To see this, let  $s \in U_{r_1}(U_{r_2}(A))$ . Then, there exists  $a' \in U_{r_2}(A)$  such that  $d(s, a') < r_1$  and there exists  $a'' \in A$  such that  $d(a', a'') < r_2$ . Then, by the triangle inequality in  $X$ ,

$$d(s, a'') \leq d(s, a') + d(a', a'') < r_1 + r_2$$

Then,  $s \in U_{r_1+r_2}(A)$  and thus  $U_{r_1}(U_{r_2}(A)) \subseteq U_{r_1+r_2}(A)$ . To use this to prove the triangle inequality in  $d_{\mathcal{H}}$ , assume that  $A, B, C \subseteq X$  and let  $b = d_{\mathcal{H}}(A, C)$ . Also let  $r > d_{\mathcal{H}}(A, B)$ ,  $s > d_{\mathcal{H}}(B, C)$ .

Then, since  $A \subseteq U_r(B)$  and  $B \subseteq U_s(C)$ , by using the fact proved above, we get

$$A \subseteq U_r(U_s(C)) \subseteq U_{r+s}(C)$$

And similarly  $C \subseteq U_{r+s}(A)$ . So,  $b \leq r + s$  and since  $r$  and  $s$  were chosen arbitrarily,

$$d_{\mathcal{H}}(A, C) \leq d_{\mathcal{H}}(A, B) + d_{\mathcal{H}}(B, C)$$

Now, let  $A$  and  $B$  be closed (so compact) subsets of  $X$  and  $d_{\mathcal{H}}(A, B) = 0$ . Then, assume that  $A \neq B$ .

So WLOG, assume that there exists  $x \in A \setminus B$ . Since  $B$  is closed, there exists  $r > 0$  such that

$$B_r(x) \cap B = \emptyset$$

This implies that  $x \notin U_r(B)$  and thus  $d_{\mathcal{H}}(A, B) \geq r > 0$ . □

This proof also shows that on the set of all subsets of  $X$ ,  $d_{\mathcal{H}}$  is a pseudometric.

Given two metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ , a map  $f : X \rightarrow Y$  is called an *isometry* if for all  $a, b \in X$ ,  $d_X(a, b) = d_Y(f(a), f(b))$ . If such an  $f$  exists, the metric spaces  $X$  and  $Y$  are said to be *isometric*. A map  $f$  is an *isometric embedding* if it is an isometry onto its image.

Now, we are ready to define the **Gromov-Hausdorff distance** between compact metric spaces, introduced by Gromov in [Gro81]. Let  $(X, d_X)$  and  $(Y, d_Y)$  be compact metric spaces. The *Gromov-Hausdorff distance* between  $X$  and  $Y$ , denoted  $d_{GH}(X, Y)$ , is defined as the infimal  $\epsilon > 0$  such that there exists a metric  $d$  on  $X \sqcup Y$  with  $d|_{X \times X} = d_X$  and  $d|_{Y \times Y} = d_Y$ , where the Hausdorff distance between  $X$  and  $Y$  as subsets of  $(X \sqcup Y, d)$  is less than  $\epsilon$ .

### 1.3 Equivalent Formulations and some Bounds of the Gromov-Hausdorff Distance

A commonly used reformulation of the Gromov-Hausdorff distance uses the notion of correspondences between metric spaces [BBI01]. Given two sets  $X$  and  $Y$ , a *correspondence* between  $X$  and  $Y$  is a set  $\mathcal{R} \subset X \times Y$  such that, for every  $x \in X$  there exists  $y \in Y$  with  $(x, y) \in \mathcal{R}$  and for every  $y \in Y$  there exists  $x \in X$  with  $(x, y) \in \mathcal{R}$ .

Given any surjective map  $f : X \rightarrow Y$ , the *correspondence associated with  $f$*  is the set  $\mathcal{R}$  given by

$$\mathcal{R} = \{(x, f(x)) : x \in X\}$$

Not every correspondence can be associated with a map in this way. However, every correspondence can be obtained using two maps, as follows:

Let  $f : Z \rightarrow X$  and  $g : Z \rightarrow Y$  be two surjective maps from a set  $Z$ . Then, we can define a correspondence  $\mathcal{R}$  by

$$\mathcal{R} = \{(f(z), g(z)) : z \in Z\}$$

Similar to the concept of the distortion of a map between metric spaces, we can define the distortion of a correspondence between metric spaces.

Let  $\mathcal{R}$  be a correspondence between the metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ . The *distortion* of  $\mathcal{R}$  is given by

$$\text{dis}\mathcal{R} \triangleq \sup\{|d_X(x, x') - d_Y(y, y')| : (x, y), (x', y') \in \mathcal{R}\}$$

**Proposition** ([BBI01]). *For any two metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ , we have that*

$$d_{\mathcal{GH}}(X, Y) = \frac{1}{2} \inf_{\mathcal{R}}(\text{dis}(\mathcal{R}))$$

*holds, where the infimum is taken over the set of all possible correspondences  $\mathcal{R}$  between  $X$  and  $Y$ .*

*Proof.* Let  $r$  be such that  $r > d_{\mathcal{GH}}(X, Y)$ . Then, we can assume that  $X$  and  $Y$  are subspaces of a metric space  $(Z, d_Z)$  and  $d_{\mathcal{H}}(X, Y) < r$  in  $Z$ . Now let

$$\mathcal{R} = \{(x, y) : x \in X, y \in Y, d_Z(x, y) < r\}$$

Since  $d_{\mathcal{H}}(X, Y) < r$  in  $Z$ , by the definition of Hausdorff distance, for every  $x \in X$  there exists  $y \in Y$  with  $d_Z(x, y) < r$  and similarly for every  $y \in Y$  there exists  $x \in X$  with  $d_Z(x, y) < r$ . So,  $\mathcal{R}$  is a correspondence between  $X$  and  $Y$ .

Now for any  $(x, y)$  and  $(x', y') \in \mathcal{R}$ , we have

$$|d_Z(x, x') - d_Z(y, y')| \leq d(x, y) + d(x', y') < 2r$$

which implies  $\text{dis}\mathcal{R} < 2r$ .

Now, we'll show that  $d_{\mathcal{GH}}(X, Y) < \frac{1}{2}\text{dis}\mathcal{R}$  for any correspondence  $\mathcal{R}$  between  $X$  and  $Y$ . Let  $\mathcal{R}$  be a correspondence with  $\text{dis}\mathcal{R} = 2r$ . Then, for any  $x \in X$  and  $y \in Y$ , let

$$d(x, y) = \inf_{(x', y') \in \mathcal{R}} \{d_X(x, x') + r + d_Y(y', y)\}$$

$d$  satisfies the triangle inequality since  $d_X$  and  $d_Y$  do. Notice that if  $x$  and  $y$  correspond to each other, then  $d(x, y) = r$ . By the definition of correspondences, for every  $x \in X$ , there will exist a  $y \in Y$  with  $d(x, y) = r$ . So,  $d_{\mathcal{H}}(X, Y) \leq r$ . So  $d$  is a pseudometric on  $X \sqcup Y$ .

By the definition of the Gromov-Hausdorff distance, we get that

$$d_{\mathcal{GH}}(X, Y) \leq r = \frac{1}{2}\text{dis}\mathcal{R}$$

□

Furthermore, it can be shown [KO99] that the Gromov-Hausdorff distance is equal to

$$d_{\mathcal{GH}}(X, Y) = \inf_{\substack{f: X \rightarrow Y \\ g: Y \rightarrow X}} \frac{1}{2} \max(\text{dis}(f), \text{dis}(g), \mathcal{C}(f, g)),$$

where

$$\mathcal{C}(f, g) \triangleq \sup_{x \in X, y \in Y} |d_X(x, g(y)) - d_Y(f(x), y)|$$

There also exist some bounds for the Gromov-Hausdorff distance in terms of  $\epsilon$ -isometries, which we will define now. First, we will need the concept of  $\epsilon$ -nets. Let  $\epsilon \geq 0$  be a real number and let  $(X, d_X)$  be a metric space. An  $\epsilon$ -net for  $(X, d_X)$  is a subset  $S \subseteq X$  such that

$$A \subseteq \bigcup_{x \in S} \mathbf{B}_\epsilon(x)$$

Now, let  $(X, d_X)$  and  $(Y, d_Y)$  be metric spaces and  $\epsilon \geq 0$  be a real number. A map  $f : X \rightarrow Y$  is called an  $\epsilon$ -isometry if  $\text{dis}(f) \leq \epsilon$  and the image of  $X$  under  $f$  is an  $\epsilon$ -net in  $Y$ . This leads to some bounds for the Gromov-Hausdorff distance in terms of  $\epsilon$  as follows [BBI01]

- If  $d_{\mathcal{GH}} < \epsilon$ , then there exists a  $2\epsilon$ -isometry from  $X$  to  $Y$ .
- If there exists an  $\epsilon$ -isometry from  $X$  to  $Y$ , then  $d_{\mathcal{GH}} < 2\epsilon$

**Proposition.** [BBI01]  $d_{\mathcal{GH}}$  is a metric on the space of isometry classes of metric spaces.

*Proof.* Clearly,  $A = B$  implies  $d_{\mathcal{GH}}(A, B) = 0$ ,  $d_{\mathcal{GH}}$  is nonnegative and  $d_{\mathcal{GH}}(A, B) = d_{\mathcal{GH}}(B, A)$ . To see the triangle inequality, for any metric spaces  $X_1, X_2, X_3$ , let  $d_{12}$  and  $d_{23}$  be metrics on  $X_1 \sqcup X_2$  and  $X_2 \sqcup X_3$ , respectively, while extending the original metrics in the spaces.

Define a distance function  $d_{13}$  between  $X_1$  and  $X_3$  by

$$d_{13}(x_1, x_3) = \inf_{x_2 \in X_2} \{d_{12}(x_1, x_2) + d_{12}(x_2, x_3)\}$$

This function combined with the metrics on  $X_1$  and  $X_3$  is a metric on  $X_1 \sqcup X_3$ . By the definition of  $d_{13}$ , we get

$$d_{\mathcal{H}}(X_1, X_3) \leq d_{\mathcal{H}}(X_1, X_2) + d_{\mathcal{H}}(X_2, X_3)$$

taken with the appropriate metrics. Taking the infimum over all possible  $d_{12}$  and  $d_{23}$ , we get

$$d_{\mathcal{GH}}(X_1, X_3) \leq d_{\mathcal{GH}}(X_1, X_2) + d_{\mathcal{GH}}(X_2, X_3)$$

All that is left to show is that  $d_{\mathcal{GH}}(X, Y) = 0$  implies  $X$  and  $Y$  are isometric, where  $X$  and  $Y$  are compact metric spaces. By the result above, we know that there exists a sequence  $f_n : X \rightarrow Y$  of maps with  $\text{dis} f_n \rightarrow 0$ . Since  $X$  is a compact metric space, it is separable and so we can fix a countable dense set  $S \subseteq X$ . Now choose a subsequence  $\{f_{n_k}\}$  of  $\{f_n\}$  such that for all  $x \in X$ ,  $\{f_{n_k}(x)\}$  converges in  $Y$ . WLOG assume that this holds for  $f_n$ .

Define a map  $f : S \rightarrow Y$  by  $f(x) = \lim f_n(x)$  for all  $x \in S$ . Now

$$|d(f_n(x), f_n(y)) - d(x, y)| \leq \text{dis} f_n \rightarrow 0$$

and so for all  $x, y \in S$ ,

$$d(f(x), f(y)) = \lim d(f_n(x), f_n(y)) = d(x, y)$$

Since  $f$  is a Lipschitz map,  $S$  is dense in  $X$ ,  $Y$  is compact and hence complete, then there exists a unique continuous map  $\tilde{f} : X \rightarrow Y$  extending  $f$ . Similarly, there also exists a distance-preserving map from  $Y$  to  $X$ . Thus,  $X$  and  $Y$  are isometric. □

This space with the Gromov-Hausdorff metric is path-connected, complete, and separable. [BBI01]

The following proposition from [Mém12] computes lower bounds to the Gromov-Hausdorff distance:

**Proposition.** *For two compact metric spaces  $X$  and  $Y$ , we have*

$$\begin{aligned} d_{\mathcal{GH}}(X, Y) &\geq \frac{1}{2} \inf_{\mathcal{R}} \sup_{(x,y) \in \mathcal{R}} |\text{ecc}_X(x) - \text{ecc}_Y(y)| \\ &\geq d_{\mathcal{H}}^{\mathbb{R}^+}(\text{ecc}_X(X) - \text{ecc}_Y(Y)) \\ &\geq \frac{1}{2} \max(|\text{diam}(X) - \text{diam}(Y)|, |\text{rad}(X) - \text{rad}(Y)|) \end{aligned}$$

## 1.4 The Modified Gromov-Hausdorff Distance and Curvature Sets

The calculation of Gromov-Hausdorff distance between finite metric spaces is very difficult and leads to NP-Hard problems. [Mém07] For computational reasons, it is easier to consider a variant introduced by Mémoli in [Mém12] called the modified Gromov-Hausdorff distance. It is defined by dropping the  $\mathcal{C}(f, g)$  term in

$$d_{\mathcal{GH}}(X, Y) = \inf_{\substack{f: X \rightarrow Y \\ g: Y \rightarrow X}} \frac{1}{2} \max(\text{dis}(f), \text{dis}(g), \mathcal{C}(f, g)),$$

So, the modified Gromov-Hausdorff distance between two metric spaces  $X$  and  $Y$  is given by

$$\hat{d}_{\mathcal{GH}}(X, Y) \triangleq \inf_{\substack{f: X \rightarrow Y \\ g: Y \rightarrow X}} \frac{1}{2} \max(\text{dis}(f), \text{dis}(g)),$$

which can be written in a decoupled way as

$$\hat{d}_{\mathcal{GH}}(X, Y) \triangleq \frac{1}{2} \max\left(\inf_{f: X \rightarrow Y} \text{dis}(f), \inf_{g: Y \rightarrow X} \text{dis}(g)\right),$$

As an example, [Mém12] calculates an explicit formula for  $\hat{d}_{\mathcal{GH}}(X, Y)$  in the case that  $X$  and  $Y$  are metric spaces both containing three points. So let  $X = \{x_1, x_2, x_3\}$  and  $Y = \{y_1, y_2, y_3\}$ . Also let  $a_1 = d_X(x_2, x_3)$ ,  $a_2 = d_X(x_1, x_3)$ ,  $a_3 = d_X(x_1, x_2)$ ,  $b_1 = d_Y(y_2, y_3)$ ,  $b_2 = d_Y(y_1, y_3)$ , and  $b_3 = d_Y(y_1, y_2)$ . Now let  $\phi: X \rightarrow Y$ .

One possibility is that  $\phi$  is a bijection. Then, define a function  $\delta$  that computes the minimal distortion over all bijections  $\phi$  between  $X$  and  $Y$ , by

$$\delta(X \leftrightarrow Y) \triangleq \min_{\pi} \max_i |a_i - b_{\pi_i}|,$$

where  $\pi$  ranges over the permutations of the set  $\{1, 2, 3\}$ .

The second possibility for the function  $\phi$  is that it maps two points from  $X$  to one point in  $Y$ , and the remaining point in  $X$  to another point in  $Y$ . In this case, first fix  $i \in \{1, 2, 3\}$  to be 1 for now. Let  $\phi$  map  $x_2$  and  $x_3$  to a point  $y \in Y$  and  $x_1$  to  $y' \neq y$ . So  $d_Y(y, y') = b_k$  for some  $k \in \{1, 2, 3\}$ . Then,



$$\begin{aligned}
 \text{dis}(\phi) &= \max(a_1, |d_X(x_1, x_2) - d_Y(y, y')|, |d_X(x_1, x_3) - d_Y(y, y')|) \\
 &= \max(a_1, |a_3 - b_k|, |a_2 - b_k|) \\
 &= \max(a_1, \max_{j \neq 1} |a_j - b_k|) \\
 &\geq \max(a_1, \min_k \max_{j \neq 1} |a_j - b_k|) \\
 &\geq \min_i \max(a_i, \min_k \max_{j \neq i} |a_j - b_k|) \\
 &= \gamma(X \rightarrow Y)
 \end{aligned}$$

And the final possibility for  $\phi$  is that it maps all three points in  $X$  to one point in  $Y$ . In this case, the distortion of  $\phi$  will be  $\text{diam}(X)$ . So, combining all three possibilities for  $\phi$ , we get

$$\inf_{\phi: X \rightarrow Y} \text{dis}(\phi) = \min(\delta(X \leftrightarrow Y), \text{diam}(X), \gamma(X \rightarrow Y))$$

Considering the maps from  $Y$  to  $X$  as well, we get

$$\hat{d}_{\mathcal{GH}}(X, Y) = \frac{1}{2} \max(\min(\delta(X \leftrightarrow Y), \text{diam}(X), \gamma(X \rightarrow Y)), \min(\delta(X \leftrightarrow Y), \text{diam}(Y), \gamma(Y \rightarrow X)))$$

Which gives the explicit formula for calculating  $\hat{d}_{\mathcal{GH}}(X, Y)$  between metric spaces with three points.

Importantly, for all compact metric spaces  $X$  and  $Y$ , we have that [Mém12]

$$d_{\mathcal{GH}}(X, Y) \geq \hat{d}_{\mathcal{GH}}(X, Y)$$

And  $\hat{d}_{\mathcal{GH}}$  is a metric on the isometry classes of compact metric spaces. Moreover, the modified Gromov-Hausdorff distance is not equal to the Gromov-Hausdorff distance in general. However, they are topologically equivalent within GH-precompact collections of metric spaces.

In [Gro81], Gromov introduces *curvature sets*, where  $\mathbf{K}_k(X)$  is the  $k$ th curvature set of a metric space  $(X, d_X)$  for  $k \in \mathbb{N}$ , defined by

$$\mathbf{K}_k(X) \triangleq \{\mathbf{D}_X(\mathbb{X}), \mathbb{X} \subset X \text{ and } |\mathbb{X}| = k\}$$

Importantly, the points in  $\mathbb{X}$  are sampled with replacement from  $X$ . As an example, consider the three point metric space with points  $a_1, a_2, a_3$ . Then, [Mém12]

$$\begin{aligned}
 \mathbf{K}_1(T) &= \{0\} \\
 \mathbf{K}_2(T) &= \left\{ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & a_1 \\ a_1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & a_2 \\ a_2 & 0 \end{pmatrix}, \begin{pmatrix} 0 & a_3 \\ a_3 & 0 \end{pmatrix} \right\} \\
 \mathbf{K}_3(T) &= \left\{ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\} \cup \bigcup_{P \in \Pi_3} \left\{ P^T \begin{pmatrix} 0 & a_1 & a_2 \\ a_1 & 0 & a_3 \\ a_2 & a_3 & 0 \end{pmatrix} P \right\} \\
 &\quad \cup \bigcup_{i=1}^3 \bigcup_{P \in \Pi_3} \left\{ P^T \begin{pmatrix} 0 & a_i & a_i \\ a_i & 0 & 0 \\ a_i & 0 & 0 \end{pmatrix} P \right\}
 \end{aligned}$$

Another simple example is when  $X = \mathbb{R}$ . In this case, we have that  $\mathbf{K}_2(X) = \mathbb{R}$ . More generally, if  $X$  is a geodesic, then  $\mathbf{K}_2(X) = [0, \text{diam}(X)]$ .

[Gro99] proves that the curvature sets are a full invariant of compact metric spaces. More precisely:

**Proposition.** *Two compact metric spaces  $X$  and  $Y$  are isometric if and only if  $\mathbf{K}_k(X) = \mathbf{K}_k(Y)$  for all  $k \in \mathbb{N}$ .*

*Proof.*  $\mathbf{K}_k(X) \subseteq \mathbf{K}_k(Y)$  implies all possible  $k$ -point subsets in  $X$  can be isometrically embedded into  $Y$ . Since  $X$  and  $Y$  are compact and this statement holds for all  $k \in \mathbb{N}$ ,  $X$  can also be isometrically embedded into  $Y$ . And similarly,  $Y$  can be isometrically embedded into  $X$ . Since both are compact,  $X$  is isometric to  $Y$ .  $\square$

In [Mém12], the author goes on to prove the *Structural Theorem*, which we will use extensively:

**Proposition.** *For all compact metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ ,*

$$\hat{d}_{\mathcal{G}\mathcal{H}} = \frac{1}{2} \sup_{k \in \mathbb{N}} d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)),$$

where  $d_{\mathcal{H}}^{\text{Sym}_k^+}$  is the Hausdorff distance in  $\text{Sym}_k^+$ , which is the set of all symmetric  $k \times k$  matrices with non-zero entries and zero diagonal, endowed with the metric

$$d_{\text{Sym}_k^+} \triangleq \max_{i,j} |a_{ij} - b_{ij}|, \text{ for } A = ((a_{ij})) \text{ and } B = ((b_{ij})) \in \text{Sym}_k^+$$

*Proof.* Let  $\eta > 0$  be such that  $\hat{d}_{\mathcal{G}\mathcal{H}} < \eta$  and let  $\phi : X \rightarrow Y, \psi : Y \rightarrow X$  be such that  $\max(\text{dis}(\phi), \text{dis}(\psi)) < 2\eta$ . Fix  $k \in \mathbb{N}$  and let  $\mathbb{X} = \{x_1, \dots, x_k\}$  be any collection of  $k$  points in  $X$ . Then,

$$d_{\text{Sym}_k^+}(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\phi(\mathbb{X}))) = \max_{i,j} |d_X(x_i, x_j) - d_Y(\phi(x_i), \phi(x_j))| \leq \text{dis}(\phi) < 2\eta$$

and similarly, if  $\mathbb{Y}$  is any collection of  $k$  points in  $Y$ ,

$$d_{\text{Sym}_k^+}(\mathbf{D}_X(\psi(\mathbb{Y})), \mathbf{D}_Y(\mathbb{Y})) < 2\eta$$

And so

$$d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)) < 2\eta$$

Letting  $\eta \rightarrow \hat{d}_{\mathcal{G}\mathcal{H}}$ , we get that

$$\frac{1}{2} d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)) \leq \hat{d}_{\mathcal{G}\mathcal{H}}$$

which holds for any  $k \in \mathbb{N}$ .

Now for the reverse inequality, assume that for all  $k \in \mathbb{N}$ ,  $d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)) < 2\eta$ . Let  $\epsilon > 0$  and  $\mathbb{X}_\epsilon = \{x_1, \dots, x_n\}$  be an  $\epsilon/2$ -net for  $X$ . Then, there must exist  $M \in \mathbf{K}_k(Y)$  such that  $M = \mathbf{D}_Y(\mathbb{Y}'_\epsilon)$ , where  $\mathbb{Y}'_\epsilon = \{y'_1, \dots, y'_n\} \in Y$  and  $d_{\text{Sym}_k^+}(\mathbf{D}_X(\mathbb{X}_\epsilon), M) < 2\eta$ .

Define a map  $\phi_\epsilon : \mathbb{X}_\epsilon \rightarrow Y$  by  $x_i \mapsto y'_i$  for all  $i$ . Then,  $\text{dis}(\phi_\epsilon) < 2\eta$  since  $d_{\text{Sym}_k^+}(\mathbf{D}_X(\mathbb{X}_\epsilon), M) < 2\eta$ . Repeating the process with the roles of  $X$  and  $Y$  reversed, we get  $\mathbb{Y}_\epsilon$ , which is an  $\epsilon/2$ -net of  $Y$  and  $\psi_\epsilon : \mathbb{Y}_\epsilon \rightarrow X$  with  $\text{dis}(\psi_\epsilon) < 2\eta$ .

Recall that for  $\{x_1, \dots, x_n\} \subseteq X$ , the Voronoi cell corresponding to  $x_i$  for each  $i$  is defined by

$$V_i = \{x \in X : d_X(x, x_i) < \min_{j \neq i} d_X(x, x_j)\}$$

Let  $V_i$  be the Voronoi cell corresponding to  $x_i$  for each  $x_i \in \mathbb{X}_\epsilon$ . Define a map  $\phi : X \rightarrow Y$  by  $x \mapsto \phi_\epsilon(x_i)$  for all  $x \in V_i$ . If  $x \in X \setminus \bigcup_i V_i$ , then let  $\phi$  send  $x$  to any  $\phi_\epsilon(x_i)$  that correspond to the closest Voronoi cells  $V_i$ .

Now for any  $x, x' \in X$ , there exist  $i, j \in \{1, 2, \dots, n\}$  with  $x \in \overline{V}_i$ ,  $\phi(x) = \phi_\epsilon(x_i)$  and  $x' \in \overline{V}_j$ ,  $\phi(x') = \phi_\epsilon(x_j)$ . So,

$$\begin{aligned}
& |d_X(x, x') - d_Y(\phi(x), \phi(x'))| \\
&= |d_X(x, x') - d_Y(\phi_\epsilon(x_i), \phi_\epsilon(x_j))| \\
&\leq |d_X(x_i, x_j) - d_Y(\phi_\epsilon(x_i), \phi_\epsilon(x_j))| + |d_X(x, x') - d_X(x_i, x_j)| \\
&\leq 2\eta + d_X(x, x_i) + d_X(x', x_j) \\
&= 2\eta + \epsilon
\end{aligned}$$

So,  $\text{dis}(\phi) < 2\eta + \epsilon$ . One can similarly construct  $\psi : Y \rightarrow X$  with  $\text{dis}(\psi) < 2\eta + \epsilon$ . Then,

$$\hat{d}_{\mathcal{G}\mathcal{H}}(X, Y) \leq \frac{1}{2} \max(\text{dis}(\phi), \text{dis}(\psi)) \leq 2\eta + \frac{\epsilon}{2}$$

Letting  $\epsilon \rightarrow 0$  and  $\eta \rightarrow \frac{1}{2} \sup_{k \in \mathbb{N}} d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y))$ , we get

$$\hat{d}_{\mathcal{G}\mathcal{H}} \leq \frac{1}{2} d_{\mathcal{H}}^{\text{Sym}_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y))$$

□

The authors remark that numerical estimation of the modified Gromov-Hausdorff distance using the structural theorem can be explored further. This is precisely what we aim to do in this paper.



## 2 Estimating the Modified Gromov-Hausdorff Distance

### 2.1 The Setup and the Expectation of the Modified Gromov-Hausdorff Distance

We will use the Structural Theorem to compute the modified GH distance between two metric spaces. First, we show that we only need to calculate the Hausdorff distance between the largest curvature sets. Assume we are given two finite metric spaces  $(X, d_x)$  and  $(Y, d_Y)$  with  $|X| = n$  and  $|Y| = m$ , where  $n, m \in \mathbb{N}$ . Assume without loss of generality that  $n \leq m$ . Then,

$$d_{\mathcal{H}}^{Sym_m^+}(\mathbf{K}_m(X), \mathbf{K}_m(Y)) \geq d_{\mathcal{H}}^{Sym_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)), \text{ where } m > k \in \mathbb{N}.$$

To see this, let  $\mathbb{X} = \{x_1, x_2, \dots, x_k\}$  and  $\mathbb{Y} = \{y_1, y_2, \dots, y_k\}$  be  $k$ -many points sampled with replacement from  $X$  and  $Y$  respectively, with a fixed ordering of points for both  $\mathbb{X}$  and  $\mathbb{Y}$ , for which

$$d_{\mathcal{H}}^{Sym_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y)) = d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y}))$$

holds. Such  $\mathbb{X}$  and  $\mathbb{Y}$  are guaranteed to exist by the definition of the Hausdorff distance. Then, define  $\mathbb{X}'$  to be  $\{x_1, x_2, \dots, x_k, x_k\}$ , so add  $x_k$  as the final element to  $\mathbb{X}$ . Then, if there exists  $\mathbb{Y}' = \{y'_1, y'_2, \dots, y'_{k+1}\}$  in  $Y$  that has  $k + 1$  points (possibly with replacement) such that

$$d^\infty(\mathbf{D}_X(\mathbb{X}'), \mathbf{D}_Y(\mathbb{Y}')) < d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y}))$$

holds, if we define  $\mathbb{Y}'' = \{y'_1, y'_2, \dots, y'_k\}$ , then by the definition of  $d^\infty$  and  $\mathbb{X}'$ , we have that

$$d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y}'')) \leq d^\infty(\mathbf{D}_X(\mathbb{X}'), \mathbf{D}_Y(\mathbb{Y}')) < d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y}))$$

This contradicts the optimality of  $\mathbb{X}$  and  $\mathbb{Y}$  corresponding to the Hausdorff distance. Thus, there is no such  $\mathbb{Y}'$ , meaning for any  $\mathbb{Y}' \subseteq Y$  with  $|\mathbb{Y}'| = k + 1$ ,

$$d^\infty(\mathbf{D}_X(\mathbb{X}'), \mathbf{D}_Y(\mathbb{Y}')) \geq d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y}))$$

Since we have that

$$d_{\mathcal{H}}^{Sym_{k+1}^+}(\mathbf{K}_{k+1}(X), \mathbf{K}_{k+1}(Y)) \geq \min_{\mathbb{Y}'} d^\infty(\mathbf{D}_X(\mathbb{X}'), \mathbf{D}_Y(\mathbb{Y}')),$$

where the minimum is taken over all possible  $k + 1$  points and their orderings taken with replacement from  $Y$ , combining the previous two equations, we get our result:

$$d_{\mathcal{H}}^{Sym_{k+1}^+}(\mathbf{K}_{k+1}(X), \mathbf{K}_{k+1}(Y)) \geq d^\infty(\mathbf{D}_X(\mathbb{X}), \mathbf{D}_Y(\mathbb{Y})) = d_{\mathcal{H}}^{Sym_k^+}(\mathbf{K}_k(X), \mathbf{K}_k(Y))$$

Now, we'll try to calculate the expected modified Gromov-Hausdorff distance for finite metric spaces. We'll do this as follows:

- We will assume that we are given two finite metric spaces  $X$  and  $Y$ , both with  $n$  points such that in each space, the distances between the  $n$  points are i.i.d. with the standard uniform distribution. This is a useful simplification to indicate if our methods to calculate the modified GH distance work well, but it is not true in general.
- We have seen that we only need to calculate the Hausdorff distance between the  $n$ -curvature sets of  $X$  and  $Y$  and not on the smaller curvature sets. However, rather than calculating this, we will instead calculate the Hausdorff distance between certain subsets of the  $n$ -curvature sets, which will ease calculations.

The subsets we refer to in the second item above is constructed by not allowing replacements when computing the curvature sets. Since  $X$  and  $Y$  both have  $n$  points, and we do not allow replacements when computing the  $n$ -curvature sets, we are essentially calculating the Hausdorff distance between the collections of all possible distance matrices of  $X$  and  $Y$ , corresponding to the orderings of the  $n$  points in  $X$  and  $Y$ .

Call the set of all distance matrices in a finite metric space  $Z$   $\mathcal{M}_Z$ . Then, we want to calculate

$$d_{\mathcal{H}}^{\infty}(\mathcal{M}_X, \mathcal{M}_Y) = \max(\max_{D_X \in \mathcal{M}_X}(\min_{D_Y \in \mathcal{M}_Y}(d^{\infty}(D_X, D_Y))), \max_{D_Y \in \mathcal{M}_Y}(\min_{D_X \in \mathcal{M}_X}(d^{\infty}(D_X, D_Y))))$$

Note that calculating  $\min_{D_Y \in \mathcal{M}_Y}(d^{\infty}(D_X, D_Y))$  for any  $D_X \in \mathcal{M}_X$  gives a lower bound for

$$\max_{D_X \in \mathcal{M}_X}(\min_{D_Y \in \mathcal{M}_Y}(d^{\infty}(D_X, D_Y)))$$

There are  $\binom{n-1}{2}$  many distances in the fixed  $D_X$ . Assuming the distances are distinct, there are  $n!$  different distance matrices  $D_Y \in \mathcal{M}_Y$ . First, let's calculate

$$\mathbb{E}[d^{\infty}(D_X, D_Y)] = \mathbb{E}\left[\max_{i,j} |D_{X_{ij}} - D_{Y_{ij}}|\right] = \mathbb{E}\left[\max_{1 \leq i \leq \binom{n-1}{2}} |X_i - Y_i|\right]$$

Here,  $X$  and  $Y \in \mathbb{R}^{\binom{n-1}{2}}$  contain the distances in  $D_X$  and  $D_Y$ . This notation is easier to work with, and we can do this because of the iid assumption.

The distribution of  $|X_i - Y_i|$  where  $X_i$  and  $Y_i$  are uniformly distributed is the triangular distribution, with the cdf

$$F(t) = 2t - t^2, \text{ for } 0 \leq t < 1$$

Thus, the distribution function of  $\max_{1 \leq i \leq \binom{n-1}{2}} |X_i - Y_i|$  is

$$F(t) = (2t - t^2)^{\binom{n-1}{2}}, \text{ for } 0 \leq t < 1$$

So the expectation becomes

$$\mathbb{E}\left[\max_{1 \leq i \leq \binom{n-1}{2}} |X_i - Y_i|\right] = \int_0^1 \left(1 - (2t - t^2)^{\binom{n-1}{2}}\right) dt$$

Applying MCT with  $f_n(t) = (2t - t^2)^n$ , this integral can be seen to approach 1 as  $n$  approaches infinity.

As a next step, let's calculate

$$\mathbb{E} \left[ \min_{1 \leq i \leq n} \max_{1 \leq j \leq \binom{n-1}{2}} |X_j - Y_j^{(i)}| \right]$$

Here,  $Y_j^{(i)}$  indicates the  $j$ 'th value in the  $i$ 'th distance matrix of  $Y$ . The cdf of the maximums were calculated already. The cdf of the minimum will then be

$$F(t) = 1 - \left(1 - (2t - t^2)^{\binom{n-1}{2}}\right)^{n!}$$

Similar to before, the expectation will be equal to

$$\int_0^1 \left[1 - \left(1 - \left(1 - (2t - t^2)^{\binom{n-1}{2}}\right)^{n!}\right)\right] dt = \int_0^1 \left(1 - (2t - t^2)^{\binom{n-1}{2}}\right)^{n!} dt$$

This integral also approaches 1 as  $n$  approaches infinity, but this is trickier to see. Since the function inside the integral is bounded by the constant function 1, which is integrable over the interval  $(0, 1)$ , if we show the convergence of the function inside as  $n$  approaches infinity, we can apply the Dominated Convergence Theorem to the integral.

So we need to show  $\lim_{n \rightarrow \infty} \left(1 - (2t - t^2)^{\binom{n-1}{2}}\right)^{n!} = 1$ . First, since for  $0 < t < 1$ ,  $0 < 2t - t^2 < 1$ , we can instead compute

$$\lim_{n \rightarrow \infty} \left(1 - u^{\binom{n-1}{2}}\right)^{n!}, \text{ where } 0 < u < 1.$$

Now,

$$\begin{aligned} \lim_{n \rightarrow \infty} \left(1 - u^{\binom{n-1}{2}}\right)^{n!} &= \lim_{n \rightarrow \infty} \exp\left(n! \log\left(1 - u^{\binom{n-1}{2}}\right)\right) \\ &= \lim_{n \rightarrow \infty} \exp\left(n! \sum_{m=1}^{\infty} \frac{\left(-u^{\binom{n-1}{2}}\right)^m}{m}\right) \end{aligned} \quad (1)$$

Note that for  $0 < x < 1$ ,

$$0 > \sum_{m=2}^{\infty} \frac{-x^m}{m} > \sum_{m=2}^{\infty} -x^m = \frac{x^2}{x-1}$$

And thus

$$(1) \geq \lim_{n \rightarrow \infty} \exp\left(-n! u^{\binom{n-1}{2}} - n! \frac{\left(u^{\binom{n-1}{2}}\right)^2}{1 - u^{\binom{n-1}{2}}}\right)$$

All that is left to show that the expression inside exp goes to 0. Now as  $n \rightarrow \infty$ ,

$$n! u^{\binom{n-1}{2}} = n! u^{\frac{n(n-1)}{2}} \leq n^n u^{\frac{n(n-1)}{2}} = \left(nu^{\frac{n-1}{2}}\right)^n \rightarrow 0$$

And finally

$$0 \leq n! \frac{\left(u^{\binom{n-1}{2}}\right)^2}{1 - u^{\binom{n-1}{2}}} \leq n! u^{\binom{n-1}{2}}$$

This proves that the limit is indeed 1. Note that this also shows that the limit of the expectation for the Hausdorff distance as the number of points approaches infinity is also 1, since that expectation is greater than the one we just calculated, and bounded from above by 1.

So, this is an indication that the Gromov-Hausdorff distance between two finite metric spaces with uniformly distributed distances and the same number of points will approach  $\frac{1}{2}$ . This also gives an expected Gromov-Hausdorff distance between two finite metric spaces, which closely matches the distances we obtained when creating the dataset, in the next section.

### 2.2 Creating the Dataset

To train a supervised learning model to learn to compute the modified GH distance between finite metric spaces, we will need a labeled dataset to train the model on. For us, this means we need to create a dataset where an element of the dataset includes a pair of distance matrices corresponding to a pair of finite metric spaces, and the approximated modified GH distance between them.

The way we achieve this is as follows:

- Randomly generate pairs of distance matrices corresponding to two metric spaces
- Randomly select  $N$  elements (a distance matrix) from the  $n$ -curvature set of both metric spaces, where  $n$  is the size of the larger metric space.  $N$  can often be chosen as 1
- Using heuristic methods, calculate the distance between the chosen distance matrices and the nearest point (in the  $l^\infty$  sense) in the corresponding  $n$ -curvature sets
- Take the maximum of the two distances computed in the last step, and divide it by 2. This gives us an approximation of the modified GH distance between the original metric spaces.

Let's now expand on these points.

#### Generating the Metric Spaces

To randomly generate a distance matrix for a metric space with  $n$  points, we generate distances by using a standard uniform distribution, and checking if the new distance satisfies the triangle inequality for all existing points and keep trying until a valid point is found.

To diversify the dataset, different distributions have been used such as the beta distribution, but this can lead to situations where the triangle inequality constraint makes it very difficult to compute new distances, and we made sure that if a new distance could not be found in a certain number of tries, the matrix was calculated from the beginning. Doing this allowed this method to work.

In addition to generating the distances this way, we also created distance matrices from graphs, by first replacing the 1 values in the adjacency matrix of the graphs with randomly generated values between 0 and 1, and then using the Floyd-Warshall algorithm to create the distance matrices. This will be explained later in the fourth chapter.

#### Selecting Distance Matrices from Curvature Sets

As we have seen, it is sufficient to calculate the Hausdorff distance between the largest curvature sets to calculate the modified GH distance. Since these sets are very large, it is infeasible to compute the Hausdorff distance between them. However, we have seen in the previous section that we can approximate



this distance by selecting random elements and computing the distance between the selected element and the nearest point in the other curvature set. This approximation will be more useful the more points the metric spaces have.

Selecting a random element from the  $n$ -curvature set of a metric space  $X$  is equivalent to sampling  $n$  points with replacement from  $X$ , fixing an ordering, and generating the distance matrix of the corresponding pseudo-metric space. This is precisely what we do to select elements of curvature sets.

### Calculating the distances

Given an element of the  $n$ -curvature set of the metric space  $X$ , our aim is to find an element of the  $n$ -curvature set of the metric space  $Y$  that minimizes the  $l^\infty$  distance between the two elements (distance matrices)

It is a very difficult task to compute this distance exactly. It is similar to the bottleneck quadratic assignment problem, where the objective function is the maximum of the absolute differences between the distance matrices, instead of the maximum of the multiplications. Given an element from the  $n$ -curvature set of  $X$ , how can we find the nearest element (or an element near enough) in the  $l^\infty$  sense in the  $n$ -curvature set of  $Y$ ? Trying to calculate all possibilities is not feasible since assuming we have  $n$  distinct points, we would have to check  $n!$  different orderings. Instead, we will use heuristic methods.

We tried using three different methods. We will first explain how each of them works, and then we will compare them to see which one performs best. First, we used a greedy algorithm that starts by randomly fixing the first point in the ordering, and selecting subsequent points by minimizing the maximum absolute distance between the relevant entries. In pseudocode, this looks like

---

#### Algorithm 1 Greedy Algorithm for Constructing Distance Matrix

---

```

1: Initialize empty list selected_points
2: Pick a random point  $p$  from  $Y$  and add to selected_points
3: Initialize best_distance =  $+\infty$ 
4: while |selected_points| <  $n$  do
5:   for each point  $q$  in  $Y$  do
6:     for each point  $a$  in selected_points do
7:       Calculate the distance  $d(q, a)$ 
8:     end for
9:     Calculate tentative_partial_matrix using the new distances  $d(q, a)$ 
10:    Calculate  $l^\infty$  distance between the corresponding entries of  $\mathbf{D}_X(\mathbb{X})$  and tentative_partial_matrix

11:    if this  $l^\infty$  distance < best_distance then
12:      Update best_distance
13:      Store  $q$  as candidate_point
14:    end if
15:  end for
16:  Add candidate_point to selected_points
17: end while
18:
19: return  $l^\infty$  distance between  $\mathbf{D}_X(\mathbb{X})$  and the distance matrix constructed using selected_points

```

---

Perhaps unsurprisingly, this algorithm produces poor results even for small values of  $n$ .

The second method is using simulated annealing, which performs much better. Simulated annealing is an optimization technique that is very useful in finding approximations to the global minimum of complicated and high dimensional functions.

The idea behind it is as follows: We start with a random solution, i.e. with random  $n$  points from  $Y$ . We also initialize the **temperature**  $T$  and the **cooling rate**. The temperature  $T$  is initialized with a large value, and a large  $T$  makes it more likely for the algorithm to accept worse solutions, meaning a solution that makes the  $l^\infty$  distance larger between the elements of curvature sets. The reason this is useful is because it allows the process to explore a larger portion of the solution space. The temperature is cooled after each iteration by multiplying it by the cooling rate, which is a value between 0 and 1. A smaller cooling rate makes the algorithm prone to being stuck at local minima, whereas a larger cooling rate has a better chance at finding the global minimum, but the process will take longer.

At each iteration, the algorithm either swaps the position of two points in the  $n$  selected points from  $Y$  with probability 0.5, otherwise replaces a point in the selected points with a random point in  $Y$ . If this leads to an improvement of the  $l^\infty$  distance between the distance matrix of the selected  $n$  points and the given distance matrix  $\mathbf{D}_X(\mathbb{X})$ , or if the statement

$$\exp\left(\frac{\text{best\_distance} - l^\infty \text{ distance}}{T}\right) > \text{random}(0, 1)$$

is true, the modification to the selected  $n$  points is accepted. In the beginning,  $T$  is large and so the algorithm looks for new, potentially better solutions. As  $T$  is decreased, it instead starts looking to improve the best solutions. It is essential to initialize  $T$  with a large value and to set the cooling rate to be large enough. In pseudocode, the algorithm looks like

---

**Algorithm 2** Simulated Annealing Algorithm for Constructing Distance Matrix
 

---

```

1: Initialize empty list selected_points
2: Pick  $n$  random points from  $Y$  and add to selected_points
3: Initialize  $T =$  some high initial value
4: Initialize best_distance =  $+\infty$ 
5: Initialize cooling_rate = some value between 0 and 1
6: while  $T >$  some threshold do
7:   Generate rand_prob = random number between 0 and 1
8:   if rand_prob  $<$  0.5 then
9:     Randomly swap two points in selected_points to create new_selected_points
10:  else
11:    Replace a random point in selected_points with a random point from  $Y$  to create
    new_selected_points
12:  end if
13:  Calculate tentative_distance_matrix using new_selected_points
14:  Calculate  $l^\infty$  distance between  $\mathbf{D}_X(\mathbb{X})$  and tentative_distance_matrix
15:  if  $l^\infty$  distance  $<$  best_distance or  $\exp((\text{best\_distance} - l^\infty \text{ distance})/T) >$  random(0, 1) then
16:    Update best_distance
17:    Update selected_points = new_selected_points
18:  end if
19:   $T = T \times$  cooling_rate
20: end while
21:
22: return  $l^\infty$  distance between  $\mathbf{D}_X(\mathbb{X})$  and the distance matrix constructed using selected_points

```

---

This method performed very well. For small  $n$ , where we can calculate the correct distance exactly via brute force, simulated annealing almost always calculates the distance correctly. In larger  $n$ , while we can not check the result directly, it consistently returns a distance value significantly better (smaller) than the one returned by the greedy algorithm.

The final method we wish to discuss is using genetic algorithms. Genetic algorithms aim to find the optimal solution by emulating natural selection and evolution on a “population” of candidate solutions. To begin, we randomly select  $n$  points from  $Y$  with replacement for  $K$  many candidate solutions. This is the first generation. From these solutions, select the half that has the lowest  $l^\infty$  distance between its distance matrix and  $\mathbf{D}_X(\mathbb{X})$  to act as parents of the next generation. These are the individuals with the highest “fitness”.

To produce the next generation, randomly select pairs of parents  $K$  times, and for each  $K$  pairs of parents, select a “crossover point” from  $n$  points. The offspring will have its points before the crossover point from one parent, and the points after the crossover point from the other parent. This method is called one-point crossover, and it is a simple way of performing crossover, or recombination. Other ways of doing this can be investigated, but this is the method we used for this task.

To introduce variation and diversity to the population, there is a small chance for a “mutation”. When each offspring is created, there is a small probability  $p_m$  for it to undergo a mutation, which replaces a random point in the offspring with a random point from  $Y$ . It is crucial for mutations to be a factor, to be able to explore a larger portion of the solution space. Without mutations, the first generation will determine the best possible solution that can be achieved. However, it is also important not to have too large of a mutation probability, since it could prevent sufficient progress in generation changes.

We continue this process for a predefined number of generations, and return the individual in the final generation with the highest fitness. In pseudocode, this algorithm is as follows

---

**Algorithm 3** Genetic Algorithm for Constructing Distance Matrix

---

```

1: Initialize population of size  $K$  with random individuals
2: Evaluate the fitness of each individual in the population
3: Set  $generation\_count = 0$ 
4: while  $generation\_count < max\_generations$  do
5:   Select parents based on fitness
6:   for each pair of parents do
7:     Select a random crossover point
8:     Swap segments before and after the point to create offspring
9:   end for
10:  for each offspring do
11:    With probability  $p_m$ , swap two points OR replace a point with a random point from  $Y$ 
12:  end for
13:  Evaluate the fitness of the offspring
14:  Select individuals for the next generation from parents and offspring
15:  Update  $best\_fitness$ 
16:  Increase  $generation\_count$  by 1
17: end while
18: return  $l^\infty$  distance between  $\mathbf{D}_X(\mathbb{X})$  and the distance matrix of the individual with the best fitness

```

---

This algorithm in its current form performs well, but not as well as the simulated annealing method shown earlier. However, there is a modification we can make to hopefully increase performance. Note that the fittest individuals in a generation never get passed directly to the next generation. Instead, they get

passed to the pool of parents, where they have a chance to be a parent to individuals in the next generation.

This presents a number of possible problems. First, it is often the case that two individuals who have high fitnesses on their own can have an offspring that has low fitness. This effectively “erases” both individuals from the population since they aren’t passed to the next generation and their offspring is not fit enough to survive to become a parent for the next generation. Another problem is that since the parents of offsprings are chosen randomly, there is some probability that a subset of the individuals with the highest fitness are not even chosen as parents, essentially wasting the progress made with those individuals.

One possible approach to tackle this problem is using what’s known as “elitism” in the genetic algorithm. This approach makes it so that a fixed number of the individuals with the highest fitness in each generation, called the “elite individuals” get passed on to the next generation directly, completely avoiding mutations and still being considered as parents for the next generation. This way, even if the elite individuals are not considered as parents, or if they are considered as parents but the offspring is not of high fitness, the elite individuals are not erased from the population forever. This also guarantees that the fitness level of the fittest individual will be nondecreasing across the generations.

With this modification, the algorithm performs much better. In fact, it is on par with the simulated annealing algorithm shown earlier. In testing both methods, we found that while both produced good results, both algorithms can outperform each other depending on the given distance matrix and the curvature set to select an element from.

In pseudocode, the genetic algorithm with the added elitism modification looks as follows

---

**Algorithm 4** Genetic Algorithm with Elitism for Constructing Distance Matrix

---

```

1: Initialize population of  $P$  randomly generated solutions
2: Evaluate the fitness of each individual in the population using  $l^\infty$  distance to  $A$ 
3: Sort the population by fitness
4: Initialize best_distance =  $+\infty$ 
5: Initialize best_individual = null
6: for each generation do
7:   Sort the population by fitness
8:   Select top  $E$  elite individuals and add them to the new population
9:   Update best_distance and best_individual if a better solution is found among elites
10:  Perform selection on the entire population of  $P$  individuals to select  $(P - E)$  parents
11:  Perform crossover on selected parents to produce  $(P - E)$  offspring
12:  for each pair of parents do
13:    Select a random crossover point
14:    Swap segments before and after the point to create offspring
15:  end for
16:  Perform mutation on offspring, excluding the elite individuals
17:  for each non-elite offspring do
18:    With probability  $m$ , swap two random points
19:  end for
20:  Replace the old population with the new population of elites and offspring
21:  Evaluate the fitness of each individual in the new population, excluding the elites
22: end for
23:
24: return  $l^\infty$  distance between  $\mathbf{D}_X(\mathbb{X})$  and the distance matrix of the individual with the best fitness

```

---

Overall, while both the genetic algorithm and simulated annealing methods performed well, we found that simulated annealing was more consistent and hence we use that algorithm in generating our dataset.



## 3 The Model

The model we will use to learn the modified GH distance between two finite metric spaces will be trained on a dataset we built using the techniques in the last chapter. This dataset is structured so that it includes the upper triangular values (to avoid duplicates and the diagonal) of the distance matrices of two metric spaces, labeled with the modified GH distance between them, computed in the last section. The goal is to see how well a neural network model can fit this problem, and how much it speeds up the process. First though, we will give a very brief refresher on neural networks.

### 3.1 Introduction to Neural Networks

A *neural network* is a supervised machine learning model that aims to build a function  $f$  that takes in an input vector  $X = (X_1, \dots, X_n)$  and predicts a response  $Y$ . The structure of a neural network is different from the other popular models with the same goal such as trees or generalized additive models.

In a regression setting, we want to predict the value of continuous *target variables*  $t$ , given the value of  $D$ -dimensional *input vectors*  $\mathbf{x}$ . If we are given a *training set* consisting of  $N$  pairs  $(\mathbf{x}_i, t_i)$ , where  $i = 1, \dots, N$ , we want our model to be able to predict the value of a *new*  $t$  corresponding to a given new  $\mathbf{x}$ .

In a linear regression model, we can take the linear combinations of *basis functions*, which are a fixed set of nonlinear functions on the input variables. These models will be linear in terms of the adjustable parameters, which makes them easy to work with. However, because of the curse of dimensionality, they are severely limited in applications where the input space has a large dimension.

#### The Structure of Neural Networks

Neural networks only fix the *number* of basis functions, but allows to learn the parameter values during training to change the basis functions themselves. As an example, a *single layer* neural network looks like

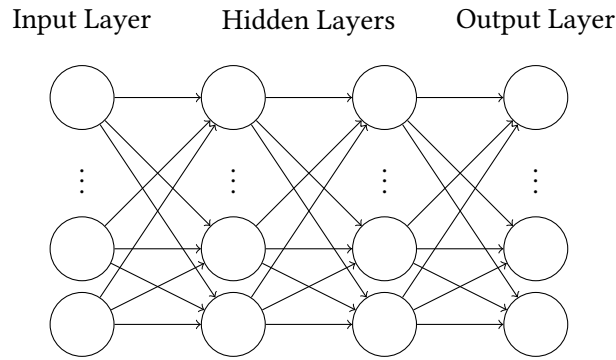
$$f(X) = \beta_0 + \sum_{i=1}^K \beta_i h_i(X),$$

where the *activations*  $h_i(X)$ ,  $i = 1, \dots, K$  in the *hidden layer* are defined as

$$h_i(X) = g(w_{i0} + \sum_{j=1}^n w_{ij} X_j),$$

where  $g$  is called an *activation function*, and is nonlinear. A popular choice for  $g$  is the *rectified linear unit*, or *ReLU*, defined as  $g(x) = x$  for  $x \geq 0$  and  $g(x) = 0$  otherwise.

So, the model first takes  $K$  linear combinations of the inputs  $X_1, \dots, X_n$ , and then applies a nonlinear activation function  $g$  to the results. Finally, another linear combination is taken *without another activation function* to get a result. Importantly, the function  $g$  must be nonlinear, or the model would become a linear model of the inputs. [Bis06]



**Figure 3.1** A typical structure of a neural network.

### Training Neural Networks

The parameters  $\beta_i$ ,  $i = 0, \dots, K$  and  $w_{10}, \dots, w_{Kn}$  are estimated from the training dataset. This is done by minimizing a loss function, and for regression tasks, the mean squared error loss is often used:

$$\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_i^n \|y_i - \hat{y}_i\|_2^2,$$

where  $\mathbf{y}$  is the *ground truth* and  $\hat{\mathbf{y}}$  is the prediction of the model. We want to minimize the loss function in order to find better predictions. This is usually done by variations of *gradient descent* to adjust the parameters of the model. For each weight (and bias), we apply

$$w_{ji}^l \leftarrow w_{ji}^l - \alpha \frac{\partial L}{\partial w_{ji}^l},$$

where  $\alpha$  is an adjustable hyperparameter called the *learning rate*.

For large training sets and large model sizes, the gradient is very expensive to compute. For this reason, variants of *stochastic gradient descent* is preferred in practice. The idea is to use only small subsets (mini-batches) of the data to compute the gradient and average them, which will approximate the true gradient well. [GBC16]

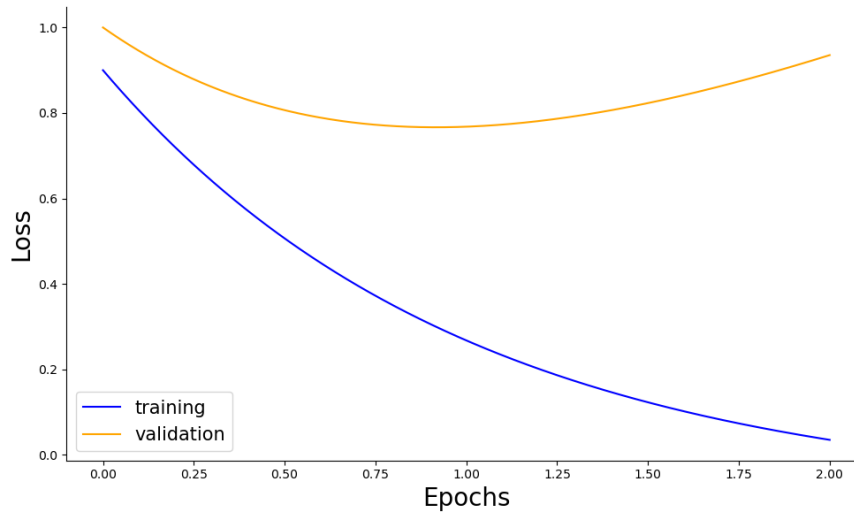
A complete pass through the training set using minibatches is called an epoch. Importantly, we also keep track of the loss function using a *validation set* in each epoch. This set is a small part of the original dataset that the model never sees during training. This way, we can see if the model *generalizes* well, and we can prevent *overfitting*.

To compute the gradients of the weights and biases, a technique called *error backpropagation*, or simply *backprop* is used, which exploits the chain rule in calculus to make the computations feasible. [GBC16]

## 3.2 The Structure of the Model

The model we use is a Siamese Neural Network model. A Siamese Neural Network model includes two identical subnetworks, which are embedding networks. The inputs (which in our case are the upper triangular values of two distance matrices) are passed through the subnetworks to obtain two vectors in the feature space. Then, a function such as the Euclidean distance is used to measure the similarity between the two vectors. First, let's talk about the model where we assume that the two metric spaces we input into the model have the same number of points.





**Figure 3.2** Image explaining overfitting/underfitting. Ideally, we want to stop training when the validation loss stops going down.

In this model, the embedding networks consist of an input layer with  $n$  neurons, corresponding to the number of distances between pairs of points in the input metric spaces, a hidden layer with  $\lfloor \frac{n}{2} \rfloor$  neurons, and an output layer with the same number of neurons. The ReLU activation function is used in between the layers. Instead of using a function like the Euclidean distance, we will concatenate the two vectors obtained from the embedding subnetworks and feed the new vector to another neural network to obtain an approximation to the modified GH distance.

This final neural network has an input layer of  $n$  neurons again, matching the dimension of the concatenated vector, made up of two  $\lfloor \frac{n}{2} \rfloor$  inputs. It again has a hidden layer of  $\lfloor \frac{n}{2} \rfloor$  neurons, and the output layer has 1 neuron. In between the layers, we again use ReLU, and finally, we use the Sigmoid function on the output.

Note that since the dataset was generated such that every distance in the metric spaces is between 0 and 1, the modified GH distance will also be between 0 and 1. This is why after concatenating the vectors in the feature space and passing them through a neural network, we have a Sigmoid activation function at the end in order to squash the values into the interval  $[0, 1]$ . Usually, the Sigmoid function is used in binary classification to interpret the outputs as probabilities, but this is not the case for us.

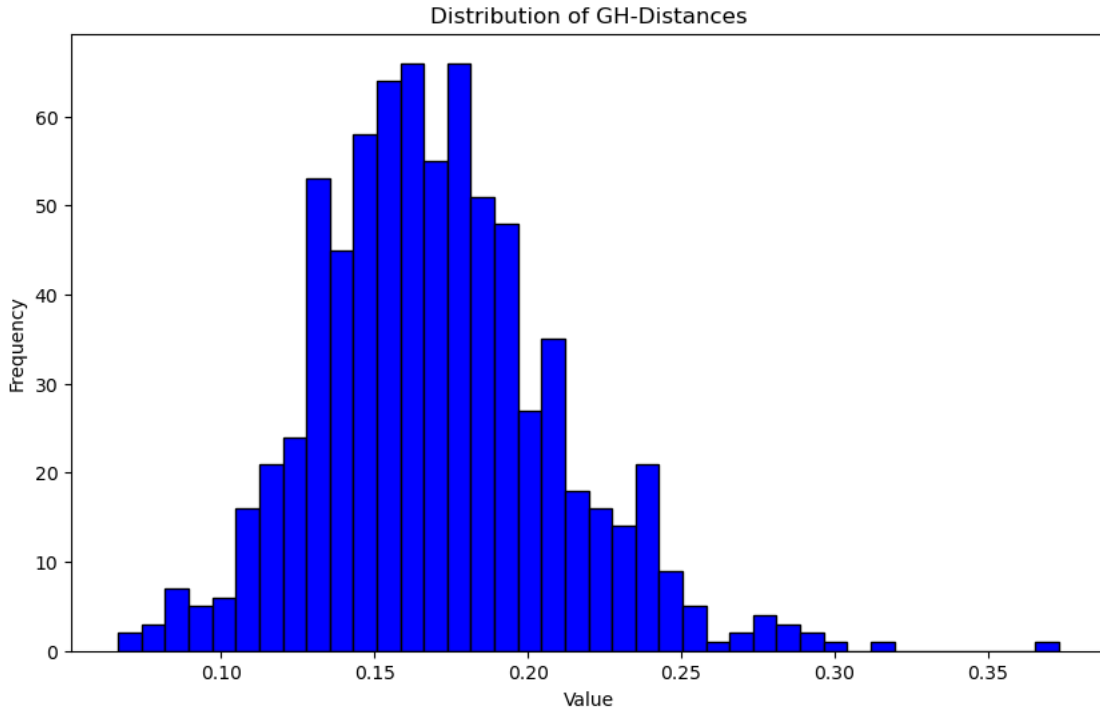
### 3.3 Training the Model

Our dataset contains 5000 datapoints of triplets  $(A, B, \hat{d}_{GH})$  where  $A$  and  $B$  are the upper triangular values of the distance matrices of the metric spaces  $X$  and  $Y$  with the same number of points, and  $\hat{d}_{GH}$  is the modified Gromov-Hausdorff distance between  $X$  and  $Y$  calculated with the methods of the last section. Figure 3.3 is the histogram showing the distribution of the calculated modified GH distances in our dataset:

To train our model on this dataset, we use the Adam optimizer [KB14] with the learning rate  $\alpha = 10^{-4}$  and a minibatch size of 32.

We use the Mean Squared Error as our loss function which, as a reminder, is defined as follows:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



**Figure 3.3** Distribution of modified GH-distances in the validation dataset

where  $y_i$  is the true value of  $\hat{d}_{GH}$  and  $\hat{y}_i$  is the predicted value of  $\hat{d}_{GH}$  by the model.

For monitoring and logging, after each epoch we log the training and validation loss to evaluate the performance and potentially for early stopping. Also, we can monitor the convergence of the losses and prevent overfitting and underfitting. Figure 3.4 shows the training and validation loss curves during training.

### 3.3.1 Different Dimensions

The Gromov-Hausdorff distance can be calculated between two metric spaces of any size. To reflect this fact, we want to build a similar model to what we have built, that can calculate the modified GH distance between metric spaces with different numbers of points. The way we achieve this is by using padding techniques.

Recall that the inputs to our model are the upper triangular values of the distance matrices of the metric spaces. If we set an upper limit  $n$  to the number of points the metric spaces can have, we can pad the input values of metric spaces with less than  $n$  points with zeros to get to a size of  $\frac{n(n-1)}{2}$ . So for a metric space with  $m < n$  points, the input will be the usual  $\frac{m(m-1)}{2}$  values followed by  $\frac{n(n-1)}{2} - \frac{m(m-1)}{2}$  many zeros.

This approach is problematic, however. The reason for this is that whenever we have a metric space with fewer points than the upper limit, it is very often the case that the vast majority of the values in the input are zeros, making it difficult for the model to learn. Our models performed poorly and did not generalize to higher dimensions. It may be the case that significantly more data points are needed to generalize properly in this case.

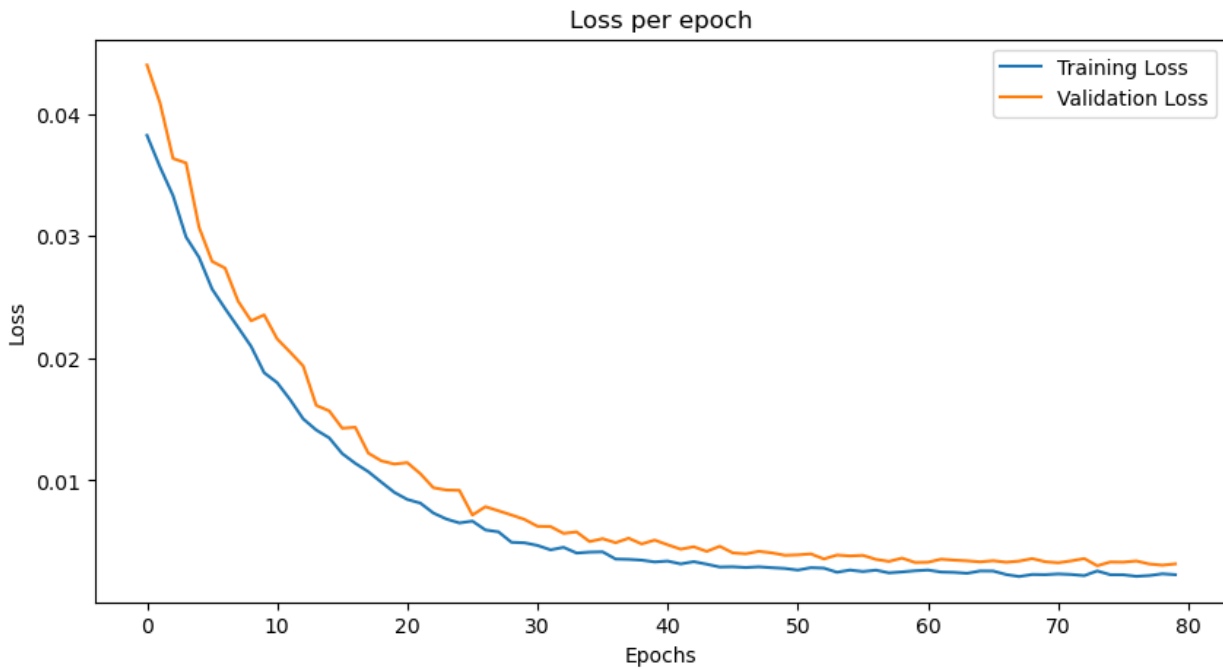


Figure 3.4 Loss curves of the model during training

### 3.3.2 Dropout and Batch Normalization

#### Batch Normalization

Batch normalization is a technique introduced in [IS15] as a technique to reduce the *internal covariate shift*, which the authors define to be “the change in the distribution of network activations due to the change in network parameters during training”. The way this is done is as follows: Given a mini-batch  $\mathcal{B}$  of size  $m$ , compute the mean  $\mu_{\mathcal{B}}$  and variance  $\sigma_{\mathcal{B}}^2$  for each feature dimension from the mini-batch

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$$

Then, normalize the activations with

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

where  $\epsilon$  is a small constant added for numerical stability. Finally, by scaling and shifting, we get

$$y_i = \gamma \hat{x}_i + \beta$$

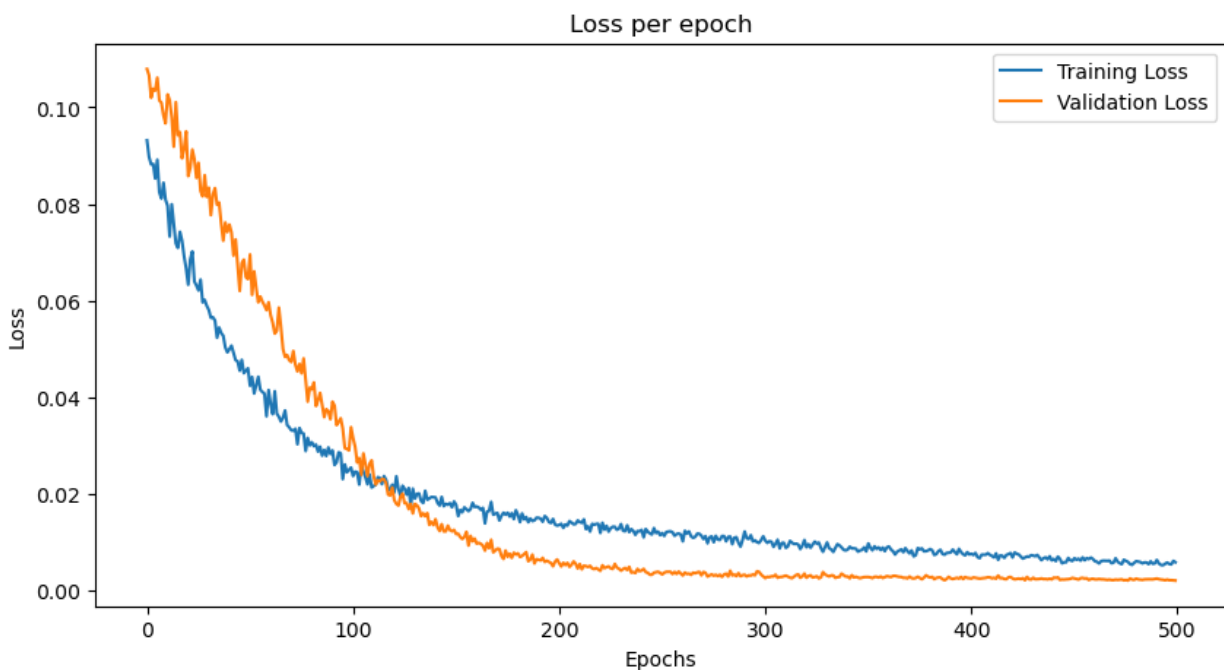
where  $\gamma$  and  $\beta$  are learnable parameters.

Adding batch normalization to the model structure improved the model’s training speed and performance. The batch normalization layers were added before the ReLU activation functions.

## Dropout

Dropout is a simple technique introduced in [Sri+14] that aims to provide regularization by making the training more difficult and hoping to lower the validation loss, thus reducing overfitting. It works by assigning a dropout probability  $p$  to each neuron ( $p$  is the same for each neuron) and “dropping out” each neuron with probability  $p$  during training at each forward/backward pass. “Dropping out” here means that neuron will be deactivated and will not be considered as part of the network, and the training will continue for the non-deactivated neurons only.

We used dropout with  $p = 0.5$  after the activation functions in our model, which improved its performance. Even though the general consensus seems to be not to use batch normalization together with dropout, this proved to be the best performing combination in our case. Figure 3.5 shows the training losses of this model.



**Figure 3.5** Loss curves of the model with batch norm and dropout layers during training

It is interesting to note that in Figure 3.5, the validation loss eventually goes below the training loss, which is normally highly unusual and unexpected. However, it is normal to see this in our case, because of the dropout layers: Dropout is only applied during *training*, and not during validation. This means that when computing the training loss, dropout is applied as usual, so on average half the neurons will be unavailable, lowering performance. However, during validation, dropout is not applied and we have the “full power” of the network. Thus, as the network gets trained more and more, it is not surprising that the validation loss gets lower than the training loss.

## 4 Using the Model to Discriminate between Different Types of Graphs

In this chapter, we want to find out if the Gromov-Hausdorff distance (and our model) can discriminate between different families of graphs. Details on how we test this will be given after a refresher on graphs.

### 4.1 A Brief Introduction to Graphs

#### 4.1.1 Basic Definitions

A *graph*  $G$  is an ordered pair  $G = (V, E)$ , where  $V$  is the set of *vertices* of  $G$  and  $E$  is a set of pairs of vertices, whose elements are called the *edges* of  $G$ . The vertices  $u$  and  $v$  of an edge  $e \in E$  are called the endpoints of  $e$ . A *weighted* graph is a graph where each edge has a weight (in  $\mathbb{R}$ ) assigned to it. A graph is *finite* if both its vertex and edge sets are finite. We will work with finite graphs.

If an edge connects two vertices, they are said to be *adjacent*. Similarly, two edges are called adjacent if they are connected to a common vertex. If the endpoints of an edge are identical, that edge is called a *loop*. Otherwise, it is called a *link*. If the graph contains no loops, it is called a *simple graph*.

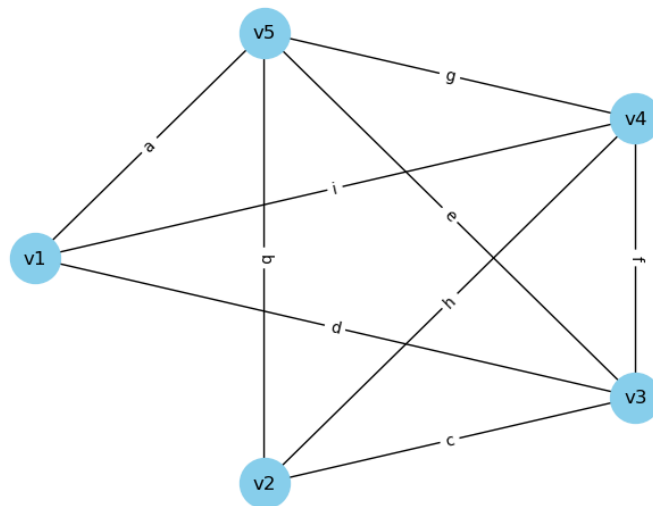


Figure 4.1 An example of a graph.

Two graphs  $G$  and  $H$  are said to be *isomorphic*, denoted  $G \cong H$ , if there exists a bijection  $f : V(G) \rightarrow V(H)$  with the property that  $u, v \in V(G)$  are adjacent if and only if  $f(u), f(v) \in V(H)$  are adjacent.

A *subgraph*  $H = (V_H, E_H)$  of a graph  $G = (V, E)$  is a graph with  $V_H \subseteq V$  and  $E_H \subseteq E$ , with the requirement that  $V_H$  includes all endpoints of  $E_H$ . If  $H \neq G$ , then  $H$  is called a *proper subgraph* of  $G$ .  $H$  is called a *spanning subgraph* of  $G$  if  $H$  is a subgraph of  $G$  and  $V_H = V$ .

### Incidence and Adjacency Matrices

The *incidence matrix* of a graph  $G$  with  $n$  vertices and  $m$  edges is an  $n \times m$  matrix  $\mathbf{M}(G)$ , where the  $(i, j)$ 'th entry of  $\mathbf{M}(G)$  signifies if the  $i$ 'th vertex is incident with the  $j$ 'th edge (1 if they are incident, 0 otherwise). As an example, the incidence matrix of the graph in Figure 4.1 is equal to

	$a$	$b$	$c$	$d$	$e$	$f$	$g$	$h$	$i$
$v_1$	1	0	0	1	0	0	0	0	1
$v_2$	0	1	1	0	0	0	0	1	0
$v_3$	0	0	1	1	1	1	0	0	0
$v_4$	0	0	0	0	0	1	1	1	1
$v_5$	1	1	0	0	1	0	1	0	0

Similarly, we can define the *adjacency matrix*  $\mathbf{A}(G)$  of a graph  $G$ . Assuming again that  $G$  has  $n$  vertices and  $m$  edges,  $\mathbf{A}(G)$  is an  $n \times n$  matrix where the  $(i, j)$ 'th entry is 1 if the  $i$ 'th and  $j$ 'th entry are joined by an edge, and is 0 otherwise. Continuing with the same example, the adjacency matrix of the graph in Figure 4.1 is equal to

	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$
$v_1$	0	0	1	1	1
$v_2$	0	0	1	1	1
$v_3$	1	1	0	1	1
$v_4$	1	1	1	0	1
$v_5$	1	1	1	1	0

### Degree of a Vertex

The *degree* of a vertex  $v$  in  $G$  is the number of edges of  $G$  that are incident with  $v$ . Any loop counts as two edges.

**Proposition.** Given a graph  $G = (V, E)$

$$\sum_{v \in V} d(v) = 2\epsilon,$$

where  $\epsilon = |E|$ .

*Proof.* Let  $\mathbf{M}$  be the incidence matrix of  $G$ . Each row of  $\mathbf{M}$  corresponds to a vertex  $v$  of  $G$  and the sum of all entries in that row is  $d(v)$ . So,  $\sum_{v \in V} d(v)$  is the sum of all entries in  $\mathbf{M}$ . However, each column of  $\mathbf{M}$  sums to 2 and there are  $\epsilon$  columns corresponding to each edge of  $G$ . Thus, this sum is also equal to  $2\epsilon$ .  $\square$

A corollary of this fact is that [Bon82] the number of vertices of odd degree is even, in any graph.

### Walks, Trails, Paths, Connectivity and Cycles

- A *walk* in a graph  $G$  is a finite sequence of edges  $\{e_1, e_2, \dots, e_n\} \subseteq E(G)$  such that there exists a sequence of vertices  $\{v_0, v_1, \dots, v_n\} \subseteq V(G)$  such that the edge  $e_i$  joins the vertices  $v_{i-1}$  to  $v_i$ .
- A *trail* is a walk with distinct edges.
- A *path* is a walk with distinct vertices (and thus edges).

In a weighted graph, the *weight of a path* is the sum of all the weights of the edges in the path.

Two vertices of a graph are said to be *connected* if there is a path connecting them. A graph  $G$  is said to be a *connected graph* if each pair of vertices are connected, and otherwise it is said to be *disconnected*. Being connected is an equivalence relation on the set of vertices  $V$  of a graph  $G$ . The equivalence classes are called the *connected components* of  $G$ . Thus,  $G$  is connected if and only if it has one connected component, and is disconnected otherwise.

A walk is *closed* if its initial and final vertices are the same. A closed trail is called a *cycle* if its first vertex is distinct from its internal vertices. A  $k$ -cycle is a cycle of length  $k$ . An *acyclic* graph is a graph that contains no cycles.

### Trees

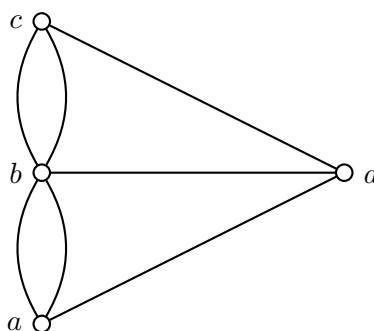
A graph  $G$  is called a *tree* if it is acyclic and connected.

Any two vertices of a tree are connected by a unique path. Conversely, if a graph  $G$  has no loops and any two vertices of  $G$  are connected by a unique path,  $G$  is a tree. The proof of this (and much more detail on trees) can be found in [Bon82].

### Eulerian and Hamiltonian Paths

One might be interested to see if, in a given graph  $G$ , we can select a vertex and starting from that vertex, cross every edge in the graph exactly once. Further, would it be possible to return to the initial vertex while crossing every edge exactly once? How about visiting every vertex exactly once? To formalize these questions, we begin with the definition of an Eulerian trail.

An *Eulerian trail* in a graph  $G$  is a trail that crosses every edge in  $G$  once and only once. If such a trail ends at the same vertex it begins in, it is called an *Eulerian circuit*. If a graph  $G$  contains an Eulerian circuit,  $G$  is said to be *eulerian*. The graph below depicts the famous seven bridges of Königsberg graph, which is not eulerian:



**Proposition.** A nonempty connected graph  $G$  contains an Euler cycle if and only if it has no vertices of odd degree.

*Proof.* Let  $G$  be an eulerian graph and  $C$  be an Euler cycle of  $G$ , whose initial and final vertex is  $v$ . Following  $C$ , every time we visit a vertex  $v'$ , we will pass through two edges incident with  $v'$ .  $C$  contains every edge in  $G$ , meaning for every edge  $u \neq v$ ,  $d(u)$  is even. Since  $C$  begins and ends at  $v$ ,  $d(v)$  is even as well,

so  $G$  can not have any vertices with odd degree.

For the other direction, assume towards a contradiction that  $G$  is a connected graph that is not eulerian, and  $G$  has no vertices of odd degree. Also assume that  $E(G) \neq \emptyset$ . Then, choose  $G$  to have as few edges as possible. Since every vertex of  $G$  has degree at least two,  $G$  must contain a closed trail. Choose such a closed trail with maximum possible length and call it  $C$ .  $C$  is not an Euler cycle since we assumed  $G$  to not be eulerian. Thus, the graph  $G \setminus E(C)$  has a connected component  $G'$  where the number of edges in  $G'$  is larger than 0.  $C$  is eulerian by construction and hence by the previous paragraph, it contains no vertices with odd degree. Since  $G$  has more edges than  $G'$ ,  $G'$  must contain an Euler cycle  $C'$ . By the connectedness of  $G$ , there exists  $v \in V(C) \cap V(C')$  and WLOG  $v$  is the beginning and end of  $C$  and  $C'$ . However,  $CC'$  is then a closed trail of  $G$  containing more edges than  $C$ , contradicting the way  $C$  was constructed. So, if a connected graph contains no vertices of odd degree, it is eulerian.  $\square$

This proposition gives a simple necessary and sufficient condition for eulerian graphs. Unfortunately, no such simple conditions are known for hamiltonian graphs, which we define now.

A *Hamiltonian path* in a graph  $G$  is a path that contains every vertex of  $G$ . A *Hamiltonian cycle* is a Hamiltonian path with the same first and last vertex. A graph is said to be *hamiltonian* if it contains a Hamiltonian cycle. There exists a simple necessary condition, whose proof can be found in [Bon82].

**Proposition.** *Let  $G = (V, E)$  be a hamiltonian graph. Then, for every proper subset  $S$  of  $V$  with  $S \neq \emptyset$ ,*

$$\omega(G \setminus S) \leq |S|$$

where  $\omega(G \setminus S)$  is the number of connected components of the graph  $G \setminus S$ .

## 4.2 The Setup

An interesting question is if the Gromov-Hausdorff distance can discriminate between different types of graphs. Of course, since it is defined as a distance between metric spaces, we would first have to convert the graphs into metric spaces, and then feed them into the Gromov-Hausdorff distance function to see the result. The way we do this is first obtaining the graphs. Since we want to see if the Gromov-Hausdorff distance can discriminate between different types of graphs, we want a dataset consisting of different families of graphs. We obtained these from the website [houseofgraphs.org](http://houseofgraphs.org) [Gra23].

The graphs provided are represented by adjacency matrices. To transform an adjacency matrix into a distance matrix representing the shortest paths between every pair of vertices, we use the Floyd-Warshall algorithm, which works as follows:

First, the algorithm initializes the diagonal of the distance matrix to be 0, and if there is no edge between the vertices  $i$  and  $j$ , the  $(i, j)$ 'th entry of the distance matrix is initialized to infinity. Otherwise, if there is an edge, it is initialized to the same value as the adjacency matrix. For unweighted graphs, this value is 1. Then, the algorithm loops through all vertices and all pairs of vertices to iteratively check if there is a shorter path than the current path, resulting in the shortest path between each pair of vertices in the distance matrix. In pseudocode, this looks like Algorithm 5.

### Randomizing the Weights and Results

As mentioned, we obtained the graphs from the website [houseofgraphs.org](http://houseofgraphs.org) [Gra23]. Given the limitations of the model we have with regards to different sizes of metric spaces we talked about in the previous section, we decided to choose all graphs with the same number of vertices. Hence, this meant that after converting the graphs to metric spaces via Algorithm 5, the metric spaces would have the same number



**Algorithm 5** Floyd-Warshall Algorithm

---

```

1:  $n \leftarrow \text{length}(\text{adjMatrix})$ 
2: for  $i = 1$  to  $n$  do
3:   for  $j = 1$  to  $n$  do
4:     if  $i = j$  then
5:        $\text{distMatrix}[i][j] \leftarrow 0$ 
6:     else if  $\text{adjMatrix}[i][j] = 0$  then
7:        $\text{distMatrix}[i][j] \leftarrow \infty$ 
8:     else
9:        $\text{distMatrix}[i][j] \leftarrow \text{adjMatrix}[i][j]$ 
10:    end if
11:  end for
12: end for
13: for  $k = 1$  to  $n$  do
14:   for  $i = 1$  to  $n$  do
15:    for  $j = 1$  to  $n$  do
16:       $\text{distMatrix}[i][j] \leftarrow \min(\text{distMatrix}[i][j], \text{distMatrix}[i][k] + \text{distMatrix}[k][j])$ 
17:    end for
18:  end for
19: end for
20:
21: return  $\text{distMatrix}$ 

```

---

of points.

To see the discriminating power of our model on different families of graphs, we chose 5 families, each containing 100 graphs. Then, within each family, we sampled random pairs of graphs to compute the modified GH distance between them (after converting them to metric spaces) via our model. We thus obtained histograms for each family showing the distributions of modified GH distances within that family. However, since Algorithm 5 uses the adjacency matrices of the original graphs to construct the distance matrices of the corresponding metric spaces, and the adjacency matrices are made up of 1's and 0's, the resulting distance matrices have very few distinct values. This makes the histograms obtained via the process we described very non-descriptive since they contain very few different values.

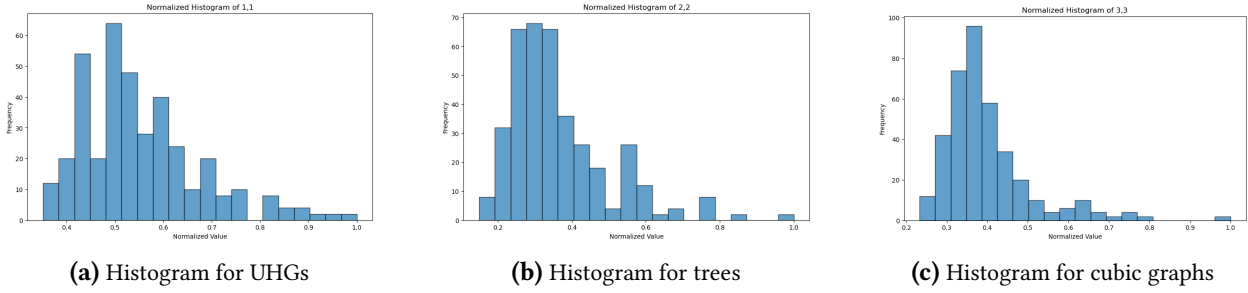
The way to remedy this is to replace the 1's in the adjacency matrices of the graphs with a random number between 0 and 1, sampled uniformly, before passing them to Algorithm 5. This way, the distance matrices do not get stuck with a few integer values, but a whole range of values. This in turn allows the histograms to be much more descriptive.

The histograms we defined so far only show the distribution of modified GH distances *within* the families of graphs. We want to see how much these histograms change when we compute the modified GH distances *between* different families of graphs. So, for each pairs of families, we randomly sample pairs of graphs, one from each family, to feed into our model. The resulting histograms will show the distribution of modified GH distances (as calculated by our model) between different types of graphs, allowing us to compare to the histograms within each family to see the differences.

The 5 different families of graphs we use are as follows:

- **Trees**, which are connected, acyclic, undirected graphs.
- **Uniquely Hamiltonian graphs**, which contain exactly one Hamiltonian cycle (a closed loop through a graph that visits each node exactly once).

#### 4 Using the Model to Discriminate between Different Types of Graphs



**Figure 4.2** Examples of histograms showing the distributions of modified GH distances within the graph families

- **Connected cubic graphs**, in which all vertices have degree three.
- **Perihamiltonian graphs**, where the graphs are non-Hamiltonian, but every edge-contracted subgraph of the graphs are Hamiltonian.
- **Platypus graphs**, which are non-Hamiltonian but every vertex-deleted subgraph contains a Hamiltonian path.

Figure 4.2 shows three examples of the histograms within graph families.

We can already see that the histograms are quite different for graphs within the families. We will now see how to make this dissimilarity more precise. Figure 4.3 shows the histograms of modified GH distances as computed by our model between all different families.

We want to see how different these histograms are from each other, so that we can determine if our model did a good job discriminating between different families of graphs. First, we'll talk about *relative entropy*. Relative entropy, also known as Kullback-Leibler (KL) divergence, is a measure of how one probability distribution differs from a second probability distribution. In the context of discrete histograms, it quantifies how one histogram differs from another.

Given two discrete probability distributions  $P$  and  $Q$ , defined over the same probability space with events  $X_1, X_2, \dots, X_n$ , the KL divergence  $D_{KL}(P||Q)$  is defined by

$$D_{KL}(P||Q) = \sum_{i=1}^n P(X_i) \log \left( \frac{P(X_i)}{Q(X_i)} \right)$$

The formula above assumes that  $P(X_i) = 0$  implies  $Q(X_i) = 0$  (absolute continuity). If there exists an  $i$  for which  $P(X_i) > 0$  but  $Q(X_i) = 0$ , then we set  $D_{KL}(P||Q) = \infty$ .

We want to characterise the dissimilarity between the histograms we generated by using the KL divergence. However, there is a problem with this approach: KL divergence is not symmetric, meaning

$$D_{KL}(P||Q) \neq D_{KL}(Q||P)$$

in general. One remedy is using the Jensen-Shannon (JS) divergence, which is symmetric. The JS divergence is defined by

$$D_{JS}(P, Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M)$$

where  $M = \frac{1}{2}(P + Q)$ .

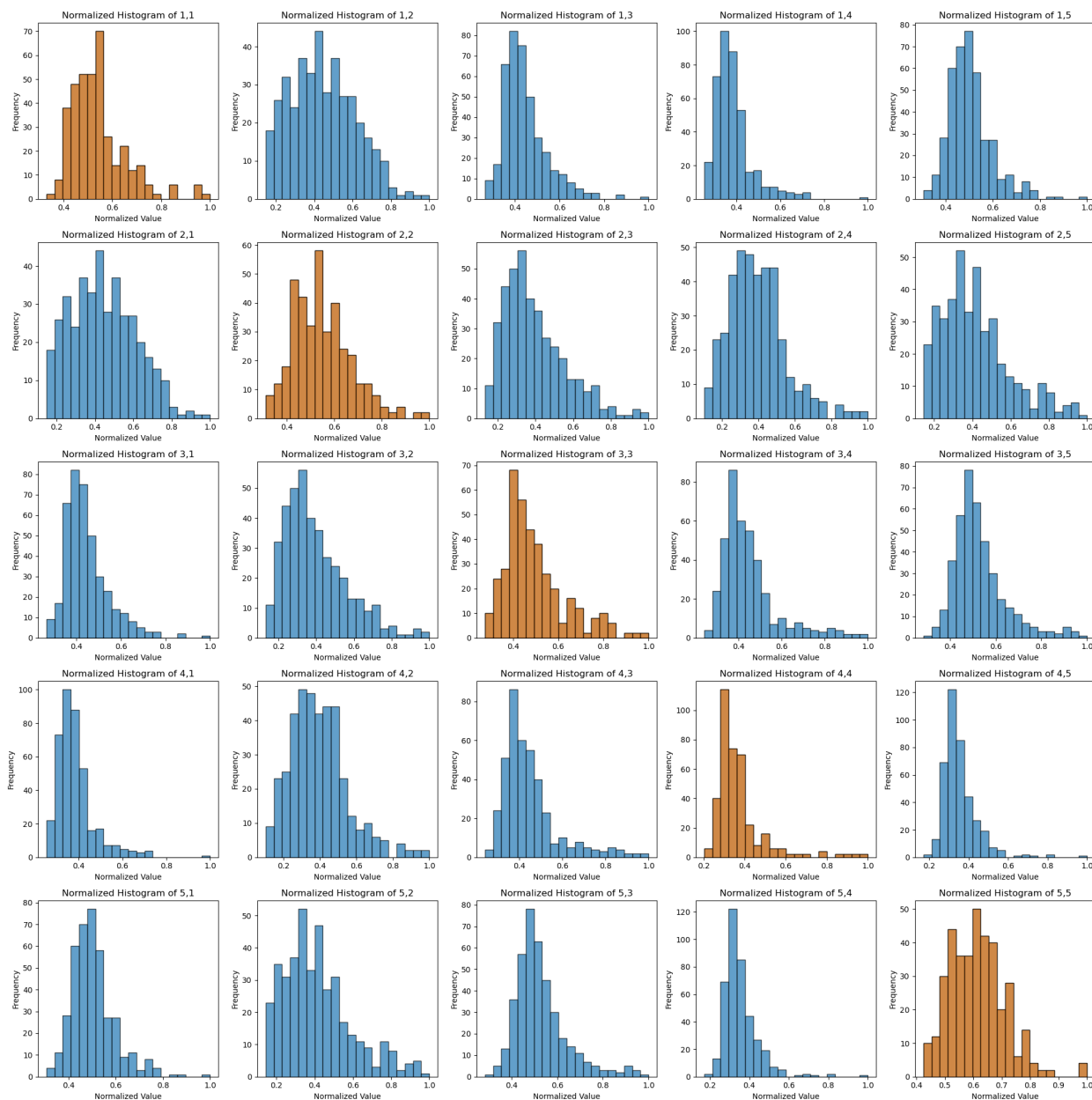


Figure 4.3 Table of histograms of all graph families and their comparisons.

Other than being symmetric, the Jensen-Shannon divergence is always finite unlike the KL divergence, which is again useful for our purposes. The JS divergence has many applications in machine learning. [Goo+14]

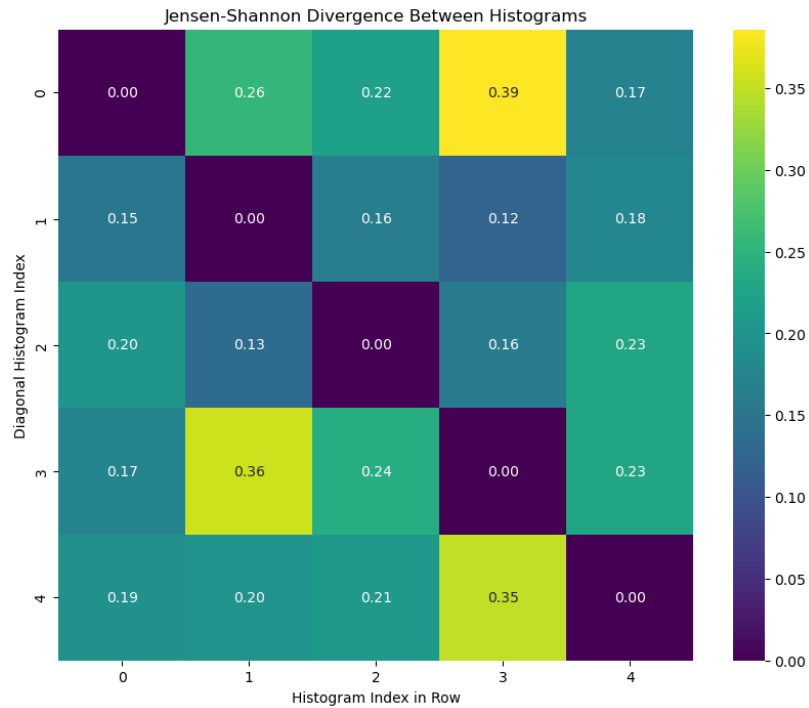
### Application to Discrete Histograms

As mentioned, both KL and JS divergence can be applied to discrete histograms by treating the histograms as probability distributions. We will use the JS divergence for the reasons outlined above. Given histograms  $H_P$  and  $H_Q$ , we first normalize them to create probability distributions  $P$  and  $Q$ . We can then use the formula for the JS divergence

$$D_{JS}(P, Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M)$$

In Figure 4.3, we computed the histograms of modified GH distance values between different families of graphs. We will now compute the JS divergence between the histograms. However, we will not compute

this divergence between every pair of histograms, but rather only between the diagonal entries and the rest of the same row. Meaning we will only compare the histogram of modified GH distances within a graph family to the modified GH distances between that graph family and other graph families.



**Figure 4.4** JS divergence between histograms. The  $(i, j)$ 'th entry represents the JS divergence between histogram  $(i, i)$  and  $(i, j)$

Clearly, unlike the table of histograms, the table in Figure 4.4 should not be symmetric, and this is indeed the case. Here, the indices of the families of graphs are

**Index 1:** Trees

**Index 2:** Uniquely Hamiltonian Graphs

**Index 3:** Connected Cubic Graphs

**Index 4:** Perihamiltonian Graphs

**Index 5:** Platypus Graphs

We can see from Figure 4.4 that the model is better at discriminating between graphs at some families than others. For example, it seems that this method could be used to discriminate between trees and other types of graphs in this list.

# List of Figures

1.1	Example of shape matching: The Gromov-Hausdorff distance provides a dissimilarity metric with good theoretical properties. The second figure shows the shapes with sampled points inside the shapes. . . . .	1
1.2	A 2D illustration of compactness, with the more common definition: A set is compact if every <i>open covering</i> of that set has a <i>finite subcovering</i> . Here, the blue circle represents the entire disk for visibility purposes, and the red dashed circles represent an open covering. .	3
1.3	Depiction of the calculation of Hausdorff distance between two randomly generated sets. The blue dashed line shows the distance from an element of Set A to the closest element of Set B, and similarly the red dashed line shows the distance from an element of Set B to the closest element of set A. The Hausdorff distance is then the maximum of all these distances.	4
3.1	A typical structure of a neural network. . . . .	24
3.2	Image explaining overfitting/underfitting. Ideally, we want to stop training when the validation loss stops going down. . . . .	25
3.3	Distribution of modified GH-distances in the validation dataset . . . . .	26
3.4	Loss curves of the model during training . . . . .	27
3.5	Loss curves of the model with batch norm and dropout layers during training . . . . .	28
4.1	An example of a graph. . . . .	29
4.2	Examples of histograms showing the distributions of modified GH distances within the graph families . . . . .	34
4.3	Table of histograms of all graph families and their comparisons. . . . .	35
4.4	JS divergence between histograms. The $(i, j)$ 'th entry represents the JS divergence between histogram $(i, i)$ and $(i, j)$ . . . . .	36



## Bibliography

- [Bel96] A. Bellaïche. “The tangent space in sub-Riemannian geometry”. In: *Sub-Riemannian Geometry*. Ed. by A. Bellaïche and J.-J. Risler. Vol. 44. Progress in Mathematics. Basel: Birkhauser, 1996, pp. 1–78.
- [Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [Bon82] J. A. Bondy. *Graph theory with applications*. 1982.
- [BBI01] D. Burago, Y. Burago, and S. Ivanov. *A Course in Metric Geometry*. Vol. 33. AMS Graduate Studies in Math. Providence: Am. Math. Soc., 2001.
- [GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [Goo+14] I. Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems 27* (2014).
- [Gra23] H. of Graphs. *House of Graphs*. Accessed: 2023-08-01. 2023. URL: <https://www.houseofgraphs.org>.
- [Gro81] M. Gromov. “Groups of polynomial growth and expanding maps (with an appendix by Jacques Tits)”. In: *Publications Mathématiques de l’IHÉS* 53 (1981), pp. 53–78.
- [Gro99] M. Gromov. *Structures métriques pour les variétés riemanniennes*. English. Trans. by S. M. Bates. Translated to English. Birkhäuser Boston, 1999, pp. xx+585. ISBN: 0-8176-3898-9.
- [IS15] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.
- [KO99] N. Kalton and M. Ostrovskii. “Distances between Banach spaces”. In: *Forum Math.* 11.1 (1999), pp. 17–48.
- [KB14] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [KS06] M. Kotani and T. Sunada. “Large deviation and the tangent cone at infinity of a crystal lattice”. In: *Mathematische Zeitschrift* 254.4 (2006), pp. 837–870.
- [Mém07] F. Mémoli. “On the use of Gromov-Hausdorff distances for shape comparison”. In: *Proceedings of Point Based Graphics 2007*. Prague, Czech Republic, 2007.
- [Mém12] F. Mémoli. “Some Properties of Gromov-Hausdorff Distances”. In: *Discrete & Computational Geometry* 48 (2012), pp. 416–440.
- [Csu] *Shape Matching using Gromov-Hausdorff distances*. Talk at Computer Vision group. CS-UC Berkeley, 2008.
- [Sri+14] N. Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.