

QUIVER THEORY, ZIGZAG HOMOLOGY AND DEEP LEARNING

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR COMPUTATIONAL AND
MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Anjan Dwaraknath

August 2020

© 2020 by Anjan Dwaraknath. All Rights Reserved.

Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.

<http://creativecommons.org/licenses/by-nc/3.0/us/>

This dissertation is online at: <http://purl.stanford.edu/ct446kb3604>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Gunnar Carlsson, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Margot Gerritsen, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Leonidas Guibas

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Topological data analysis deals with the study of data using techniques from algebraic topology. It has emerged as a popular and powerful technique in studying data in recent times. In this dissertation, we will introduce a new perspective on algorithms for topological data analysis (TDA). The new framework will allow us to generalize existing algorithms to new problems and also provide novel methods to parallelize the algorithms to compute zigzag persistent homology. We will first introduce how quiver representations can be used to reduce the problem of computing persistent and zigzag homology to that of computing a canonical form similar to the eigenvalue decomposition or the Jordan normal form. Then we will describe an algorithm to compute the canonical form of any type A quiver representation. We will show how the algorithm can be expressed as a sequence of matrix factorizations and matrix passing steps performed on the graph underlying the quiver representation. We will conclude by looking at an example of how TDA can be used along with deep learning, specifically we will show how zigzag homology performs better than persistent homology in the case of the MNIST dataset.

Acknowledgments

I would like to thank my adviser Gunnar Carlsson for the many discussions we had on various topics. More specifically I would like to acknowledge that it was his initial idea of using block elimination for zigzag that put me on the journey to discover the matrix passing algorithm.

I would also like to thank Bradley Nelson, as he was part of the many discussions over which we fine tuned the details of the algorithm. I specifically want to thank him for helping me understand the trick to convert pivot matrices from one type to another, which proved useful for the divide and conquer algorithm.

I would like to thank Leo Guibas for being part of my reading committee, but also for teaching the course "Geometric and Topological Data Analysis". This was my first course in the field and provided a very good foundation to learn things on my own later.

I would like to thank Margot Gerritsen for being part of my reading committee and also for all the support over the years. Finding an adviser can be one of the toughest parts of completing a PhD and Margot made that process a little easier for me.

I would like to thank Rafe Mazzeo for being the oral examiner at my defense. Additionally I would also like to thank Michael Saunders for chairing the defense.

I would like to acknowledge my funding sources, without which this PhD would be impossible - the Stanford Graduate Fellowship, specifically the Burt and Deedee McMurtry fellowship. I would like to thank ICME for allowing me to fund myself through teaching short courses.

Finally I would like to thank friends and family without whom the long journey would have felt even longer. I want to thank my late mother Anuradha and my father Dwarak, who have always supported and encouraged me to pursue my passion. I want to thank my brother Rajat, for always being there to bounce ideas with.

ICME was a very welcoming place and I would like to thank Nolan, Ron, Lan, Victor, Nurbek, Jordi and the rest of the ICME folk who made me feel part of the community.

Contents

Abstract	iv
Acknowledgments	v
1 Introduction	1
1.1 Contributions	3
2 Persistent and Zigzag Homology	4
2.1 Homology	4
2.1.1 Simplicial Complex	4
2.1.2 Vector Spaces	5
2.1.3 The Boundary Operator and the Chain Complex	5
2.1.4 How linear algebra models homology	7
2.1.5 Betti numbers	10
2.1.6 Category Theory Formulation	10
2.2 Persistent Homology	12
2.3 Zigzag Homology	15
3 Quiver Representations	17
3.1 Chain Complex to Quiver Representation	17
3.1.1 Canonical form	18
3.1.2 Gabriel’s theorem	18
3.2 Interpreting the Canonical form	19
3.2.1 Homology	19
3.2.2 Persistent homology	20
3.2.3 Zigzag homology	20
3.3 Change of basis	20
3.3.1 Matrix factorization form	21
3.3.2 Matrix passing algorithm	25

4	Quiver Algorithms	28
4.1	Linear algebra tools	28
4.1.1	Notations and definitions	28
4.1.2	Triangular Factorizations	31
4.1.3	Shape Commutation relations	35
4.2	Algorithm for Persistent type quivers	38
4.3	Algorithm for Zigzag quivers	40
4.3.1	General Sequential Quiver Algorithm	43
4.4	Divide and Conquer parallel algorithm for zigzag quivers	44
4.5	Numerical Examples	47
4.5.1	Subsets and Rips Complexes	47
4.5.2	Bivariate Nerve	48
4.5.3	Sierpinski Triangle	49
5	Applications of Zigzag Homology for Deep Learning	51
5.1	Neural Networks	51
5.1.1	Neural Network Architecture	51
5.1.2	Neural Network Training	52
5.2	Application of Persistence and Zigzag Homology to MNIST	53
5.2.1	Featurization	55
5.2.2	Neural Network Model	55
5.3	Experiments and Results	56
5.4	Discussion	58
	Bibliography	59

List of Figures

2.1	Action of the boundary operator on a linear combination of two triangle	7
2.2	The green cycle can be considered to be a boundary of a vector, but there is no vector whose boundary would produce the second cycle.	8
4.1	Notation for different matrices, along with pictorial symbols	31
4.2	Pictorial representation of the shape commutation relationship in Proposition 4.1.3.1	36
4.3	The factorization to use for the first sweep.	43
4.4	The commutation to use for the second sweep.	44
4.5	Time to compute zigzag homology of a zigzag diagram computed on subsamples of a noisy circle. Normal noise with variance 0.1 is added to points sampled from a unit circle. 200 points are contained in each subsample, and $r = 0.35$. The horizontal axis indicates the number of samples used in the diagram. Left: time to compute zigzag homology for both BATS and Dionysus. Right: Speedup seen using BATS instead of Dionysus. At the right hand side, BATS is over 600x faster.	47
4.6	Left: persistence diagram for Rips filtration on 200 points sampled from the unit circle. The unoptimized reduction algorithm runs in 20 seconds in BATS. Right: An approximate persistence diagram created using the discrete Morozov zigzag construction in [11] using the suggested parameters. The zigzag computation takes approximately 0.5 seconds in BATS. While the birth and death times are not identical, both diagrams qualitatively display the same information, namely a single connected component and a robust H_1 class, agreeing with the homology of the circle.	48
4.7	Zigzag barcodes of bivariate Nerve diagram on 5 covers of 500 points on the unit circle. Covers are computed by selecting 20 random landmarks. Left: each point is assigned to closest 2 landmarks. Right: each point is assigned to closest 3 landmarks. Note that both diagrams have single long bars in dimensions 0 and 1, agreeing with the homology of the circle.	49
4.8	Persistence barcode showing 4 iterations of the sequence in Equation (4.22).	50
5.1	A MNIST digit and the graph implied by its pixels.	54

5.2	The sequence of images in the filtration and the corresponding barcode	54
5.3	Table summarizing the performance of models	56
5.4	Loss graph and confusion matrix of the best model	57
5.5	Qualitative results - Some examples that were correctly classified	57

Chapter 1

Introduction

Topological data analysis emerged as an initially unlikely field which tried to apply highly abstract concepts from algebraic topology to the field of data analysis and machine learning. The very development of the subject required the cooperation of pure and applied mathematicians although the initial developments were mainly from the pure side. The nature of such an origin implied that most of the literature, including the ones that describe the algorithms, were written from a pure mathematician's perspective. This meant that most of the algorithms were algorithms of principle, i.e. they were nothing more than just a proof by construction. The situation improved and efficient algorithms were developed and implemented. Even though most of the algorithms could be expressed easily using linear algebra primitives, the literature required a lot of theoretical background to fully understand why they worked. Paradoxically, once you understood the background, most of the algorithms ended up being very specific and written as a sequence of row and column operations on various matrices which were usually messy and unfriendly to the beginner.

On the other hand in the much older field of numerical linear algebra, there had been a long history of studying various algorithms, and an efficient way to express them had evolved. Most results of algorithms were expressed as matrix factorizations and all the intermediate steps usually involving various row and column operations were also expressed as mini-factorizations. This not only allowed easier analysis of the algorithms, but also brought some method to the madness of actually implementing them. Algorithms were no longer a sequence of unintelligible operations on a matrix, they could be expressed cleanly as a sequence of factorizations to finally obtain a final factorization.

The goal of this thesis is to bring some of that order to the algorithms of topological data analysis. The spur to undertake such a task occurred when I was trying to study the algorithm used to compute zigzag homology. An auxiliary goal of this dissertation is to show the potential uses of zigzag homology and its advantages over persistence homology. However there were not many implementations of the algorithm to compute zigzag homology and the one that did exist was

very specific and verbose. Further investigation revealed that both persistence and zigzag homology can be cleanly cast as a matrix factorization problem. This was not surprising in hindsight as all the operations in the algorithm were linear algebraic and there were theoretical results that effectively said the same thing, however in the much more abstract language of quivers and quiver representations. Once this realization was made, I attempted to search for an algorithm in the style of numerical linear algebra of composing small factorizations to build the final one. Lo and behold such an algorithm could be found. This resulted in a new framework of algorithms that put both zigzag and persistence homology in the same setting. It allows you to generalize to problems that have not been explored before, such as the cases with non-inclusion maps. It also allowed the development of novel parallel algorithms which provided upto 600x improvements over existing methods. Crucially it also brought the pure and applied sides closer together. For example it was common for algebraic topologists to think about computing homology first and then computing persistence or zigzag later, now such a perspective can be translated directly to the algorithms side as well. Thus it will allow pure and applied sides to work more closely together in the development of new algorithms. On the other side of things, many insights from numerical linear algebra such as efficient pivoting to reduce fill-in and block Schur complement approaches can be easily applied to improve speed of the algorithms.

Another benefit of this new approach is that it can be expressed purely using linear algebra and one of the goals of this dissertation is to have it be accessible to someone with only linear algebra knowledge and almost no background knowledge in algebraic topology. To this end the content of this thesis is expressed in matrix language and analogies to the eigenvalue decomposition and Jordan decomposition are made. This will hopefully open the door to this subject to more people in the future.

As mentioned earlier, an auxiliary goal of this thesis is to demonstrate the potential uses of zigzag homology. Persistence homology has seen widespread use, but due to lack of general algorithms and strong understanding of zigzag homology (both of which this thesis aims to improve), it has not been used to its full potential. As a proof of concept, I study the application of using topological data analysis to solve the quintessential machine learning problem of recognizing MNIST digits. The central idea is to use TDA to generate features on which a neural network can be trained. The principle advantage of this approach is that the classifier can generalize to images of digits that are much different in size and style. This is achieved through the use of topological features to recognize digits and not individual pixel values. Another conclusion of these experiments is that features from zigzag homology perform better than persistence homology indicating that zigzag homology can be more expressive.

The structure of the thesis is as follows, Chapter 2 provides a good primer to the concepts of persistence and zigzag homology, all relevant concepts are introduced but no algorithm is discussed. Chapter 3 goes on to introduce the quiver framework in which all the algorithms will be specified.

Chapter 4 describes the algorithms and its various components in detail. Finally Chapter 5 discusses the experiments on MNIST to show an example of how zigzag homology does better.

1.1 Contributions

The main contributions of this dissertation is found in the algorithms of chapter 4 and experiments in chapter 5. The numerical experiments in chapter 4 specifically therips complex and the bi-variate nerve were performed by my collaborator Bradley Nelson, for our paper [2]. The Sierpinski triangle experiment was joint work. Some of that work can also be found in his dissertation [9]. My contributions to those experiments are the design of the underlying algorithm and the implementation of the algorithm used. The material in chapter 3 was also written for the paper [2] jointly with Brad.

Chapter 2

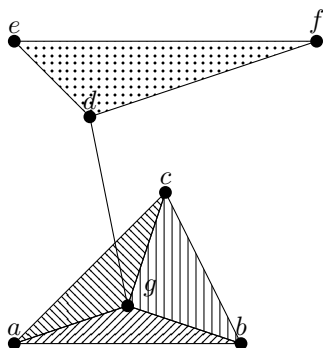
Persistent and Zigzag Homology

2.1 Homology

2.1.1 Simplicial Complex

The main mathematical object that we will be using in talking about homology is the simplicial complex. The simplicial complex can be considered to be a generalization of the graph. The graph can be defined in the following way, it consists of a set of vertices V and a set of edges E containing pairs of vertices. This definition can be extended by adding another set of higher order objects, namely triples of edges. Similar to how we can add an edge between two vertices from the vertex set, we can add a 2-simplex consisting of 3 edges from the edge set, with the constraint that the three edges form a triangle.

Let us look at the example to illustrate the idea. Here we have a simplicial complex with 7 vertices also called 0-simplices, a, b, c, d, e, f, g . We then add the edges or 1-simplices $(a, b), (b, c), \dots$ marked by solid lines. We can also add the triangles $(a, b, c), (a, b, g), \dots$ marked by the various hatched shadings. We can extend the notion of 2-simplices to even higher order simplices and the resultant



object is known as a simplicial complex. We can now formally define the simplicial complex as follows.

Definition 2.1.1.1. *A simplicial complex X is defined as a tuple (S, K) , where S is a set of abstract vertices (also called 0-simplices) and K is a set of simplices, where each simplex is defined as a subset of S . The set K has to satisfy the following condition*

$$\forall \sigma \in K \quad \tau \subset \sigma \implies \tau \in K$$

The simplex τ is called a face of the simplex σ and the above condition states that a simplex can be part of the simplicial complex if and only if all its faces are also included.

The above definition is sometimes considered to be the definition of an abstract simplicial complex since we do not require any of the simplices to be embedded in any space. In this thesis whenever we refer to simplicial complex we will always mean the abstract version.

2.1.2 Vector Spaces

We will now bring some Linear Algebra into the picture. We are going to construct a vector space for each dimension of simplices in the simplicial complex. Thus there will be a vector space V_0 with all the vertices as formal basis vectors. The coefficients will be from some field F . The dimension of the vector space will therefore be the number of vertices. Similarly we will have a vector space V_1 of all edges, V_2 of all 2-simplices etc. These are just formal vector spaces as of now and we will arrange them in the following diagram

$$0 \longleftarrow V_0 \longleftarrow V_1 \longleftarrow \cdots \longleftarrow V_k \longleftarrow \cdots$$

We have added no real content by this construction as of yet, but once we add linear transformations on each of the directed edges above, it will become more meaningful.

2.1.3 The Boundary Operator and the Chain Complex

For graphs one can construct the incidence matrix that records each edge with a row containing a 1 and -1 at the incident vertices. The incidence matrix is not just a data structure to hold the content of the graph but also has a specific function as a linear transformation. Applying it to a vector of node values produces a vector of edge values containing the differences in the node values incident to each edge. The boundary matrix is the extension of this concept to the simplicial complex. The boundary operator ∂_k is a linear transformation that takes a vector from V_k to V_{k-1} . If we define the action of ∂_k on each of the basis vectors, by linearity we can extend it to the entire vector space. We will use the angle bracket notation to denote simplices, for example $\langle a \rangle$ and $\langle b \rangle$ are individual vertices, $\langle ab \rangle$ is an edge and $\langle abc \rangle$ is a 2-simplex. This is different from using sets to denote

a simplex, as we need order information as well. So let us use a modified version of the simplicial complex.

Definition 2.1.3.1. *An ordered simplicial complex is similar to a simplicial complex, but each simplex is no longer a set, but is now a tuple of abstract vertices. They can be denoted in the following way*

$$\sigma = \langle v_1, v_2, \dots, v_k \rangle$$

The order is useful to describe the orientation of the simplex, which will allow us to define the boundary operator

By definition the action of the boundary operator on individual vertices is set to 0. The boundary of the edges can be thought of as the vertices that are incident to it. So action of ∂_1 on edges of the form $\langle ab \rangle$ produces $\langle b \rangle - \langle a \rangle$, similar to incidence matrices. The action of boundary operator ∂_2 on 2-simplices will produce the a linear combination of the edges that border it. Again the signs will be chosen to reflect the orientation of the 2-simplex. $\partial_3 \langle abc \rangle = \langle bc \rangle - \langle ac \rangle + \langle ab \rangle$.

Definition 2.1.3.2. *The general formula for the action of the boundary operator on any basis vector in the simplicial complex is as follows.*

$$\partial_k \sigma = \sum_{i=0}^k (-1)^i \langle v_0, v_1, \dots, \hat{v}_i, \dots, v_k \rangle$$

The main property of the boundary operator is the following result

Lemma 2.1.3.3. $\partial_k \partial_{k+1} = 0$

The set of vector spaces and the linear transformations together form the chain complex. This is going to be the algebraic object that we construct from the simplicial complex to encode all the information from the simplicial complex.

Definition 2.1.3.4. *A Chain complex C is defined as a sequence of vector spaces $\{V_i\}$ for $i = 0, 1, \dots$ along with linear transformations $\partial_i : V_i \rightarrow V_{i-1}$ between them.*

$$0 \longleftarrow V_0 \xleftarrow{\partial_1} V_1 \xleftarrow{\partial_2} \dots \xleftarrow{\partial_k} V_k \xleftarrow{\partial_{k+1}} \dots$$

The linear transformations satisfy the following property

$$\partial_k \partial_{k+1} = 0$$

2.1.4 How linear algebra models homology

Let us spend some time to understand the utility of the boundary operator by studying its action on linear combinations of simplices. Although there is no restriction to do so, we will look at linear combination of simplices that form contiguous regions in the complex. We will also set all the coefficients to either 1 or 0 in such linear combinations. Consider the example in Figure 2.1.

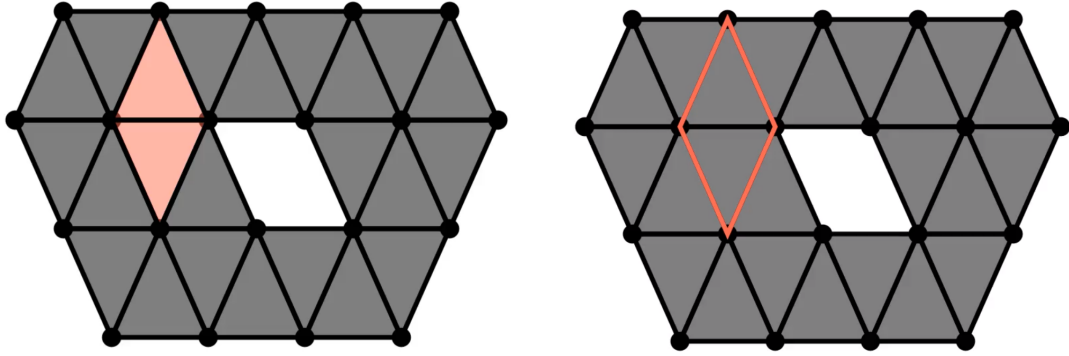


Figure 2.1: Action of the boundary operator on a linear combination of two triangle

As we can see, the action of the boundary operator on a vector representing a contiguous region, will produce the vector representing the geometric boundary of that region. This is of course only true if the coefficients for all the basis vectors are equal. Thus due to linearity, the action of the boundary operator is meaningfully extended to linear combinations with some restrictions.

In the example, we see that the boundary produced is a cycle and algebraically this manifests as the fact that the action of the boundary operator on it is 0. We will thus make it the formal definition.

Definition 2.1.4.1. *A vector $\sigma \in V_k$, representing a simplex, is a cycle if the action of the appropriate boundary operator on it produces 0*

$$\partial_k \sigma = 0$$

In other words it belongs in the kernel of the boundary operator.

$$\sigma \in \ker \partial_k$$

The next step is to notice that there are two types of cycles, some cycles can be represented as the boundary of a vector from V_2 . In other words we can find a region represented by a linear combination of 2-simplices whose boundary produces the cycle in question. On the other hand there

are cycles that can never be expressed as the boundary of a region, because the cycle encloses a “hole” in the simplicial complex. This is the key fact that will later allow us to identify topological holes in the space. Let us look at the example in Figure 2.2.

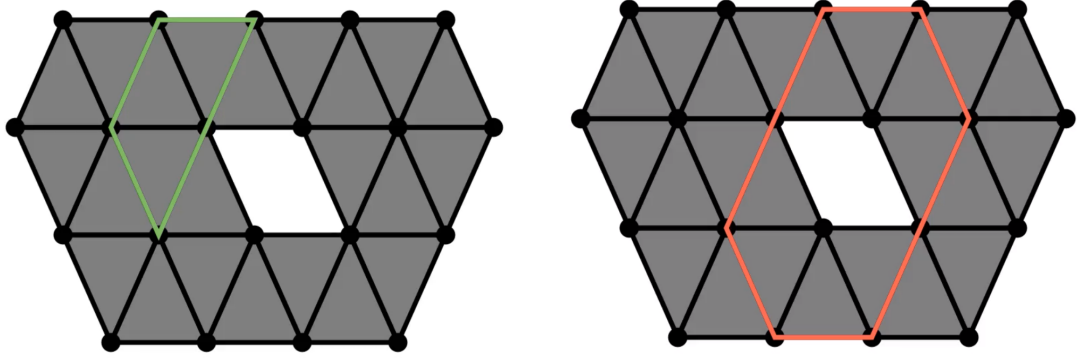


Figure 2.2: The green cycle can be considered to be a boundary of a vector, but there is no vector whose boundary would produce the second cycle.

In the above figure, the green cycle is a boundary because there exists a vector, in this case a linear combination of three triangles whose boundary would be that cycle. On the other hand if you consider the cycle in the second figure, there does not exist any such linear combination of triangles. If you just considered all the triangles enclosed by that cycle and try to apply the boundary operator on it, it would produce a boundary that would include this cycle but would also include the inner boundary surrounding the hole.

Our goal eventually is to count the number of such holes in our simplicial complex. We know that if a hole is present there will be cycles which are not boundaries, but if there are multiple such holes how do we distinguish them and count them. Given a single hole, there are multiple cycles enclosing it, all of them being non-boundaries, yet all such cycles are somehow equivalent since if we “fill in” the enclosed hole, they will all become boundaries leaving the cycles enclosing other holes as is.

Thus there are two types of cycles. qualitatively there are the cycles that enclose holes and those that do not. The way we precisely distinguish them is by asking if the cycle is actually a boundary of some vector in a higher order vector space. In other words whether the cycle is present in the image of the boundary operator. Since $\delta_k \delta_{k+1} = 0$ We know that all boundaries are cycles, but crucially all cycles are not boundaries and its these non-boundary cycles that are the ones that enclose holes and capture topological information.

The next step is to use this information to count such holes. We cannot just count non boundary cycles as there are many different such cycles that enclose the same hole, yet they are all equivalent in some sense. We will make this notion of equivalency formal and create a partition for the set of

all non boundary cycles.

To reiterate, we have a formal vector space for each dimension of simplex. These spaces are connected by the boundary operators. the kernel of these operators form the cycles. The image of these operators form boundaries and all boundaries are cycles. The cycles which are not boundaries are the ones enclosing topological features, but many such cycles enclose the same feature.

The way we can determine if two such cycles are equivalent is if we can get from one such cycle to another by just adding boundaries. Since boundaries cannot enclose any new features the new cycle must be enclosing the same hole. Thus by adding any element of the kernel you will not leave the equivalency class. Formally two vectors $v_1 \in V_1$ and $v_2 \in V_1$ are equivalent if we can express one of them as follows

$$v_1 = v_2 + u \quad u \in \delta V_2$$

We can verify that the above relation forms an equivalence relation, thus partitioning the set of non-boundary cycles into ones that characterize unique topological features.

The next step is to realize that the partition itself forms a vector space in its own right, with each class being a single vector. The way you take linear combinations of these "equivalence class" vectors is through representatives, the choice of which is irrelevant.

Lets say you have two classes, notated as follows, $[v_1]$ and $[v_2]$, here v_1 and v_2 are the representatives and the square bracket denotes that we are refering to the classes.

$$a[v_1] + b[v_2] = [av_1 + bv_2]$$

We can verify that this is consistent by replacing the classes with arbitrary representatives

$$a(v_1 + w_1) + b(v_2 + w_2) = av_1 + bw_2 + (aw_1 + bw_2) = av_1 + bw_2 + w_3 = [av_1 + bw_2]$$

Here if $w_1 \in \partial_2 V_2$ and $w_2 \in \partial_2 V_2$ then so will $w_3 \in \partial_2 V_2$

Let us denote equivalent cycles as homologous, thus this vector space of equivalency classes characterize the homology of the space.

We can generalize the process described above to any two vector spaces.

Definition 2.1.4.2. Consider two vector spaces V and W , we can define the quotient of two such spaces $Q = V/W$ as follows, Q is a vector space of equivalence classes defined by the following equivalence relation between $u \in V$ and $v \in V$

$$u \sim v \text{ if } u = v + w \quad w \in W$$

This notion of quotient spaces allows us to define the Homology vector space as follows

Definition 2.1.4.3. *The homology vector spaces given a chain complex C is defined as follows*

$$H_k(C) = \ker \partial_k / \text{im } \partial_{k+1}$$

2.1.5 Betti numbers

In the previous section we looked at how we can see intuitively that the equivalence relation induced by the boundaries capture the notion of homologous cycles. Once we have homology we can now see how we can count holes algebraically. They are just the dimension of the homology vector spaces that were constructed using the quotient operation. This is because each independent dimension would represent different equivalence class of cycles. These counts of holes of various dimensions are known as Betti numbers

Definition 2.1.5.1. *Betti numbers are defined as the dimensionality of the homology vector spaces*

$$\beta_k = \dim H_k(C)$$

Thus we have seen how we can go from a combinatorial object, namely the simplicial complex to an algebraic object - the chain complex. We then demonstrated how to use linear algebra with the chain complex to obtain topological information - the Betti numbers. In the coming sections we will build upon this basic pipeline and build more complicated structures that would be more useful in practical settings.

2.1.6 Category Theory Formulation

In order to fully understand the potential of the homology construction, we need to introduce some category theory language. Lets quickly introduce categories and functors.

Definition 2.1.6.1. *A category C consists of the following*

1. *A class of objects $ob(C)$*
2. *A class $hom_C(a, b)$ of morphisms between source object a and target object b . Each morphism can be denoted as follows $f : a \rightarrow b$*
3. *for every three objects a, b and c , a binary operation $hom_C(a, b) \times hom_C(b, c) \rightarrow hom_C(a, c)$ called composition of morphisms; the composition of $f : a \rightarrow b$ and $g : b \rightarrow c$ is written as $g \circ f$ or gf .*

Thus a category consists of objects of a certain kind and morphisms between them. Let us consider an example, the category of ordered simplicial complexes **SComp**. In order to complete the category we need to define a morphism between simplicial complexes - simplicial maps.

Definition 2.1.6.2. A simplicial map between two ordered simplicial complexes $X = (S_X, K_X)$ and $Y = (S_Y, K_Y)$, is defined as a map on the vertices $f : S_X \rightarrow S_Y$, which takes simplices to simplices,

$$\langle v_1, v_2, \dots, v_k \rangle \in K_X \implies \langle f(v_1), f(v_2), \dots, f(v_k) \rangle \in K_Y$$

Thus equipped with simplicial maps, we can now consider the set of simplicial complexes a category.

Proposition 2.1.6.3. **SComp** is category with simplicial complexes as objects and simplicial maps as morphisms.

When we construct a new mathematical object from an existing one, for instance when we constructed the chain complex from the simplicial complex, or the homology vector spaces from the chain complex, we are transforming an object from one category to another and if such a transformation satisfies certain properties, such as transforming morphisms in a consistent manner, we call them a *functor*.

Definition 2.1.6.4. Let C and D be categories. A functor F from C to D is a mapping that

1. Associates to each object X in C an object $F(X)$ in D ,
2. Associates to each morphism $f : X \rightarrow Y$ in C a morphism $F(f) : F(X) \rightarrow F(Y)$ in D such that the following two conditions hold:

- (a) $F(\text{id}_X) = \text{id}_{F(X)}$ for every object X in C ,
- (b) $F(g \circ f) = F(g) \circ F(f)$ for all morphisms $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ in C .

That is, functors must preserve identity morphisms and composition of morphisms.

Before we define a functor from the category of simplicial complexes to chain complexes, we need to make chain complexes a category as well. We need to define a morphism between chain complexes - the chain map.

Definition 2.1.6.5. A chain map between two chain complexes C and D is defined as a sequence of maps (linear transformations) $\{F_i\}$ for $i = 0, 1, \dots$, one for each dimension in the chain complex, that commutes with the boundary operators according to the following diagram

$$\begin{array}{ccccccc} 0 & \longleftarrow & V_0^C & \xleftarrow{\partial_1^C} & V_1^C & \xleftarrow{\partial_2^C} & \dots \xleftarrow{\partial_k^C} V_k^C \xleftarrow{\partial_{k+1}^C} \dots \\ & & \downarrow F_0 & & \downarrow F_1 & & \downarrow F_k \\ 0 & \longleftarrow & V_0^D & \xleftarrow{\partial_1^D} & V_1^D & \xleftarrow{\partial_2^D} & \dots \xleftarrow{\partial_k^D} V_k^D \xleftarrow{\partial_{k+1}^D} \dots \end{array}$$

Chain complexes together with chain maps form the category **CComp**.

Proposition 2.1.6.6. *CComp is category with chain complexes as objects and chain maps as morphisms.*

Proposition 2.1.6.7. *The chain map construction is a functor from SComp to CComp*

This means that we can extend simplicial maps between simplicial complexes to chain maps between chain complexes.

Proposition 2.1.6.8. *The homology construction H_K is a functor from SComp to Vect*

It turns out that the homology construction is also a functor. It takes objects from the category of chain complexes to the category of vectors spaces. There is one such functor for each k , thus H_k is a different functor for each value of k . In order to understand the implications of the above statement, let us consider two chain complexes C and D with a chain map between them. The chain map is defined by maps $\{F_i\}$. We can compute the individual homology as follows,

$$H_k(C) = \ker \partial_{k+1}^C / \text{im } \partial_k^C$$

$$H_k(D) = \ker \partial_{k+1}^D / \text{im } \partial_k^D$$

Since the homology operation behaves as a functor, we can also derive maps between the corresponding homology vector spaces as follows

$$H_K(F_k) : H_k(C) \rightarrow H_k(D)$$

These maps are referred to as the induced maps on homology. Since the class of categories themselves form a category with functors as the morphism, we can use the associative property to compose the two functors.

$$\mathbf{SComp} \rightarrow \mathbf{CComp} \rightarrow \mathbf{Vect}$$

This means that we can lift simplicial maps all the way upto homology. In the next two sections we will see methods to generate such simplicial maps and study the utility of the objects constructed when you lift them upto homology.

2.2 Persistent Homology

In the previous sections we saw how you can define the homology of a simplicial complex. The Betti numbers describe useful topological invariants, but how do we create these simplicial complexes in the first place. One source of simplicial complexes are point clouds from datasets.

Given a set of points in some d -dimensional space, one could do the following construction. Construct a ball of radius ϵ around each point, add an edge between two points if their respective

balls overlap. If three edges form a triangle, we can stipulate that we add the 2-simplex composed of those three points. Similarly we add higher order simplices if all its faces have been added. In this way we can construct a simplicial complex out of the point cloud of a dataset.

Definition 2.2.0.1. *A Rips complex of radius ϵ on a set of points $\{x_i\}$ for $i = 0, 1, \dots, n-1$ is constructed as follows*

1. *Let the set of 0-simplices be the points of the dataset, $S = \{x_i | i = 0, 1, \dots, n-1\}$*
2. *Construct the 1-skeleton $\tilde{K} = \{< x_i, x_j > | B_\epsilon(x_i) \cap B_\epsilon(x_j) \text{ is non-empty}\}$*
3. *Take the clique closure of \tilde{K} to obtain K , this is done by including any simplex in K whose edges are contained in K .*

The constructed simplicial complex (S, K) is the Rips complex for radius ϵ

The idea behind this construction is that we can now obtain Betti numbers that would characterize some topological invariants in our dataset, but how do we choose the ϵ ? It would seem that if we choose either a too small or too big ϵ we would miss some topological features or add noise to it. The choice of the correct ϵ is therefore messy and can depend heavily on the various scale parameters of the dataset.

One solution to this issue is to avoid the question of choosing a single ϵ altogether, instead we can actually sweep a range of values and study the combined output. In fact we don't have to think of the output as a sequence of Betti numbers, we can actually derive more continuity information. We can map the features discovered for one ϵ with features discovered in the next ϵ . Some features may not exist and new features may be created, but we can track them across various values of ϵ . Thus each feature will be born at some ϵ and will die at another ϵ .

This form of the output is more richer than just a sequence of Betti numbers. Each feature is represented as a bar, an interval in ϵ space, signifying the duration of existence of a topological feature. The set of such bars $\{(b_i, d_i)\}$ is called the persistence barcode and it represents the persistence homology of the point cloud. This construction not only avoids the problem of choosing a single ϵ it also provides a very robust construction to apply on real world data. We can use the length of the bars to judge how important a topological feature is. We can study how small scale features connect to the large scale features. As it turns out this construction, although motivated through completely practical considerations, is a well defined mathematical object with clean theorems regarding its existence and uniqueness.

We can generalize further, the method we used to generate a sequence of simplicial complexes, namely of trying different values of ϵ , is by no means the only approach. This particular construction is called the Rips complex. It takes as input a point cloud, a sequence of values for ϵ and outputs a simplicial complex with each simplex being annotated with an ϵ value. The annotated value dictates at which ϵ value was the simplex created. This annotated simplicial complex is called a filtration

and the only constraint is that the annotated values of the faces of a simplex should be less than or equal to the annotation value of the simplex itself. This is to guarantee that you do not try to add a simplex before adding its boundary.

Definition 2.2.0.2. A filtration f on a simplicial complex $X = (S, K)$ is defined as a map $f : K \rightarrow \mathbb{R}$ with the following constraint,

$$\sigma \subset \tau \implies f(\sigma) \leq f(\tau)$$

Given such a filtration one can choose a finite set of values from \mathbb{R} and build simplicial complexes with inclusion maps between them.

Definition 2.2.0.3. A filtered complex F is a sequence of simplicial complexes $\{X_i\}$ obtained from a filtration f , a simplicial complex $X = (S, K)$ and a finite set of points $t_i \in \mathbb{R}$, $i = 0, \dots, n-1$

$$X_i = (S, \{\sigma \in K : f(\sigma) \leq t_i\})$$

Due to the nature of the above construction, there are natural inclusion maps between successive simplicial complexes.

$$\phi_i : X_i \hookrightarrow X_{i+1}$$

These maps can be extended to the chain complexes built upon the simplicial complexes to produce a chain map. Using the homology functor, we get induced maps between the homology vector spaces.

$$\hat{\phi}_i : H_k(X_i) \rightarrow H_k(X_{i+1})$$

These vector spaces can now be arranged in its own diagram.

Definition 2.2.0.4. A persistence complex is a sequence of vector spaces, with the following arrow structure

$$H_k(X_0) \xrightarrow{\hat{\phi}_0} H_k(X_1) \xrightarrow{\hat{\phi}_1} \dots \xrightarrow{\hat{\phi}_j} H_k(X_{j+1}) \xrightarrow{\hat{\phi}_{j+1}} \dots$$

It is the output of applying the homology functor on a filtered complex or any sequence of chain complexes with chain maps in one direction.

The persistence complex can also be considered as a functor from the category of integers \mathbf{Int} with the $r \leq s$ relation as the morphism, to the category of vector spaces \mathbf{Vec}

The persistence complex contains the barcode information discussed earlier. In the next few chapters we will see how to extract the barcodes from this object.

2.3 Zigzag Homology

We saw in the previous section how a filtration could be used to produce a filtered complex. It consisted of a sequence of simplicial complexes with maps between them, but all the maps were inclusion maps and were in the same direction. After applying the homology functor, we obtain a diagram of homology vector spaces called the persistence complex. The arrow directions in this diagram also pointed in one direction, but for a barcode to exist this is not necessary.

Thus if you have arbitrary chain maps between a sequence of simplicial complexes, where arrow direction can change arbitrarily, we can still apply the homology functor to obtain a linear diagram of homology vector spaces. So long as the diagram is linear, irrespective of the arrow directions, there is always going to be a barcode representation. The proof of this result will be discussed later in the thesis.

One such diagram we could construct is the following,

$$X_0 \xrightarrow{\phi_0} X_1 \xleftarrow{\phi_1} X_2 \xrightarrow{\phi_2} X_3 \xleftarrow{\phi_3} \dots$$

Since the arrow directions alternate, such a diagram is called a zigzag diagram. Usually the maps between the chain complexes are assumed to be inclusion but they need not be for the barcode to exist. In this thesis we will see general algorithms that work even when the maps are derived from non-inclusion maps. After applying the homology functor, we obtain the zigzag complex, with induced maps between the homology vector spaces in alternating direction.

Definition 2.3.0.1. *The zigzag complex is defined as a sequence of vector spaces with the following arrow structure*

$$H_k(X_0) \xrightarrow{\hat{\phi}_0} H_k(X_1) \xleftarrow{\hat{\phi}_1} H_k(X_2) \xrightarrow{\hat{\phi}_2} H_k(X_3) \xleftarrow{\hat{\phi}_3} \dots$$

The zigzag complex encapsulates the zigzag barcodes, similar to the persistence complex. The goal of the next few chapters is to understand how we go from these diagrams to barcodes and efficient algorithms to compute them.

The above was an algebraic look at what zigzag persistence is, but where do they arise in practice. In the previous section we saw how we can use a filtration to get a persistence complex, we used a sequence of increasing filtration values and looked at the simplicial complexes generated by the sub-level sets for those filtration values. This methodology guaranteed inclusion and uni-directional maps. We can also generate zigzag diagrams using a filtration. Instead of using a sequence of values, we instead use a set of overlapping intervals on the real line, that form a cover. Each interval then corresponds to a sub-complex, provided that the filtration has the right properties. We can now generate a diagram using the intersection of these sub-complexes to glue the spaces into a single zigzag diagram.

$$X_0 \xleftarrow{\phi_1} X_0 \cap X_1 \xrightarrow{\phi_2} X_1 \xleftarrow{\phi_3} X_1 \cap X_2 \xrightarrow{\phi_4} X_2 \xleftarrow{\phi_5} X_2 \cap X_3 \xrightarrow{\phi_6} \dots$$

We do not need a filtration to generate such a diagram, any cover over the simplicial complex can also be used to generate such a diagram. We can also use unions instead of intersections to generate a different style of zigzag diagram.

$$X_0 \xrightarrow{\phi_1} X_0 \cup X_1 \xleftarrow{\phi_2} X_1 \xrightarrow{\phi_3} X_1 \cup X_2 \xleftarrow{\phi_4} X_2 \xrightarrow{\phi_5} X_2 \cup X_3 \xrightarrow{\phi_6} \dots$$

This kind of diagram appears in situations where we may have different undersamplings of a large dataset and we wish to relate the features discovered in one under sampled dataset with those found in the others. Thus long bars in this scenario indicate that multiple under samplings have reproduced this particular feature, indicating that it is significant.

Chapter 3

Quiver Representations

In the previous chapter we saw the motivations for the construction of Persistence and Zigzag homology, but we did not deal with the task of actually computing it in detail. There are various algorithms in the literature for various sub cases, in particular the reduction algorithm for persistence and the right filtration algorithm for zigzag persistence. In this thesis we wish to present a unified framework for all such algorithms that is not only easy to deal with and extend for newer situations but also provides us opportunities to parallelize it.

Before we describe the algorithms, we will first describe the framework in which the algorithms are encoded, namely quiver representations. Expressing persistence and zigzag persistence in quiver language is ofcourse not new, but the full potential for creating algorithms with it has not been explored before.

3.1 Chain Complex to Quiver Representation

We have looked at the chain complexes in the previous sections and saw that they are the primary algebraic object that can be used to compute our topological invariants. In this chapter we are going to look at a more general version of that object as that version will prove to be more useful in the future. In particular, both persistence complexes and zigzag complexes would be an instance of it. We are going to relax the constraints on how the vector spaces are obtained and what linear transformations appear on each arrow.

We will instead consider a general directed graph, with finite vector spaces on each node that can be of various dimensions. We will remove any restrictions on the linear transformations and instead assume that there can be any arbitrary linear transformation represented by a matrix on each edge. The only constraint being that the matrix dimensions match the source and destination vector spaces. The generic object we have constructed is called a quiver representation. The chain complex is a specific quiver representation with added structure such as the fact there are boundary

operators on each edge. The formal definition of a quiver representation is as follows

Definition 3.1.0.1. *A quiver representation is a directed graph $\mathcal{Q}(V, E)$ where every vertex $v_i \in V$ has an associated vector space V_i over a common field \mathbb{F} , and each directed edge $(v_i, v_j) \in E$ has an associated \mathbb{F} -linear transformation $A_{i,j} : V_i \rightarrow V_j$.*

3.1.1 Canonical form

The object we have defined can be completely specified by the directed graph and the entries of the matrices on each edge. This specification however assumes a particular basis for each of the vector spaces and in general a different basis will produce different matrices. Thus the specific element values are somewhat arbitrary as one can perform a change of basis operation and although the underlying object is the same it will have different numerical values.

This allows us to make an analogy with the eigenvalue and the Jordan decomposition. These are similar scenarios but with just one matrix. Here as well the actual numerical value of the elements has to do with the choice of the basis and is not intrinsic to the mathematical object. In both those cases there exists a change of basis to a more natural basis that reduces the matrix into a very simple form, namely either a diagonal matrix or a matrix in Jordan form. A natural question is whether quiver representations also have such a canonical form and if so can we compute them efficiently. It turns out that the answer is yes, but only for certain graphs. This question was answered by Pierre Gabriel in 1972 when it arose in a different context regarding Lie Algebras.

3.1.2 Gabriel's theorem

Theorem 3.1.2.1. *Gabriel (1972). A quiver is of finite type if it has only finitely many isomorphism classes of indecomposable representations. Gabriel classified all quivers of finite type, and also their indecomposable representations. More precisely, Gabriel's theorem states that*

1. *A connected quiver is of finite type if and only if its underlying graph (when the directions of the arrows are ignored) is one of the ADE Dynkin diagrams: A_n, D_n, E_6, E_7, E_8*
2. *The indecomposable representations are in a one-to-one correspondence with the positive roots of the root system of the Dynkin diagram.*

The summary of the result is as follows, the canonical form of the quiver representation is guaranteed to exist for any set of matrices on the edges only if the graphs are one of the so called ADE Dynkin diagrams. They are usually labelled A_n, D_n, E_6, E_7, E_8 . The E graphs are specific finite graphs, but the A and D graphs are families, we however will only care about the type A quivers as these are the only ones that will end up being useful for persistence and zigzag homology.

The type A quivers can be described as a general line graph of any length. The arrow directions can be assigned in anyway and it will still be considered a type A quiver. Each type A quiver

can be decomposed into a formal sum of indecomposable representations. The indecomposable representations of these quivers are known as interval indecomposables [3, 5, 10], and have the form

$$I[b, d] = \cdots \longrightarrow 0 \longrightarrow \mathbb{F} \longrightarrow \cdots \longrightarrow \mathbb{F} \longrightarrow 0 \longrightarrow \cdots$$

where b denotes the first index at which a copy of \mathbb{F} appears, and d denotes the final index where \mathbb{F} appears, all vector spaces with index $i \in [b, d]$ also have a copy of \mathbb{F} , with identity maps along all edges connecting two copies of \mathbb{F} and zero maps along all other edges. In other words, any quiver of type A_n is isomorphic to the direct sum of these indecomposables

$$Q \cong \bigoplus_i I[b_i, d_i]$$

As a convention, we will use the lexicographical (total) order on \mathbb{Z}^2 when ordering interval indecomposables, using the parameters b, d . These interval indecomposables are the "barcode" of the quiver, they represent what is invariant, when the basis is changed. For persistence quivers, these actually end up being the persistence barcodes

3.2 Interpreting the Canonical form

We can now interpret the objects of TDA, using quiver representations

3.2.1 Homology

We can start with the chain complex, we saw earlier that homology information can be derived from taking quotients of the various kernel and image spaces in the chain complex. However there is another way to obtain the homology - by considering it as a quiver representation and looking at its canonical form

Proposition 3.2.1.1. *The chain complex is a type A quiver representation*

$$0 \longleftarrow V_0 \xleftarrow{\partial_1} V_1 \xleftarrow{\partial_2} \cdots \xleftarrow{\partial_k} V_k \xleftarrow{\partial_{k+1}} \cdots$$

The interval indecomposables of the chain complex have very specific structure due to the following property satisfied by the boundary operator.

$$\partial_k \partial_{k+1} = 0$$

There cannot be a bar whose length is greater than 2. This is because if such a basis vector existed which represented a bar of length greater than 2, then applying the boundary operator twice would

not kill it, thus contradicting the central property of boundary operators. Thus we can represent the chain complex as follows

$$C \cong \bigoplus_i I[b_i, b_i] \oplus \bigoplus_i I[b_i, b_i + 1]$$

Here the first component, which corresponds to the length 1 bars actually correspond to the homology of the chain complex.

3.2.2 Persistent homology

We can also treat the persistence complex as a quiver representation.

Proposition 3.2.2.1. *The persistence complex is a type A quiver representation.*

$$H_k(X_0) \xrightarrow{\hat{\phi}_0} H_k(X_1) \xrightarrow{\hat{\phi}_1} \dots \xrightarrow{\hat{\phi}_j} H_k(X_{j+1}) \xrightarrow{\hat{\phi}_{j+1}} \dots$$

The persistence quiver will thus have a canonical form and the interval indecomposables will be the persistence barcodes that were discussed earlier in the previous chapter. Each basis vector in the canonical form represents an element in the homology that survives as long as dictated by the length of the indecomposable. Thus one approach to computing the barcodes is to compute the canonical form of the associated persistence quiver.

3.2.3 Zigzag homology

Changing arrow directions does not change the type of the quiver, so a zigzag complex is also a type A quiver.

Proposition 3.2.3.1. *The zigzag complex is a type A quiver representation*

$$H_k(X_0) \xrightarrow{\hat{\phi}_0} H_k(X_1) \xleftarrow{\hat{\phi}_1} H_k(X_2) \xrightarrow{\hat{\phi}_2} H_k(X_3) \xleftarrow{\hat{\phi}_3} \dots$$

Similar to the persistence case, we can compute zigzag barcodes by computing the canonical form of the zigzag quiver.

3.3 Change of basis

In order to compute the canonical form of type A quivers, one needs to find the correct change of basis transformations that will leave the matrices in the correct form. In order to understand the change of basis transformations in the context of quiver representations, we will look at the matrix algebra involved more closely. Let us look at standard change of basis for a single matrix first,

Change of Basis : Suppose that we have vector spaces V and W with bases $B^V = \{b_i^V\}$ and $B^W = \{b_i^W\}$ respectively and a linear transformation T represented by a matrix A in bases B^V and B^W . That is the coefficients $A[j] = Tb_j^V$. Now, suppose that we have new bases $C^V = \{c_i^V\}$ and $C^W = \{c_i^W\}$, where $c_i^* = U^*b_i^*$. Then if we wish to write T in terms of bases C^V and C^W , we can write a new matrix

$$\hat{A} = (U^W)^{-1}AU^V \quad (3.1)$$

The matrix U^V first maps coefficients in C^V to coefficients in B^V , then the linear transformation T is applied, mapping to coefficients in B^W , which are then mapped to coefficients in C^W via $(U^W)^{-1}$.

3.3.1 Matrix factorization form

We will now focus on classification of type A_n quivers, which appear for both persistent and zigzag homology.

Definition 3.3.1.1. *The companion matrix of a quiver representation \mathcal{Q} is the block matrix which has non-zero blocks in the non-zero entries of the adjacency matrix of the underlying directed graph, where blocks are filled by the linear transformations along the corresponding edges. By necessity, the size of the i -th block must be the dimension of V_i in the quiver.*

These matrices act on the vector space $V = \bigoplus_i V_i$ by sending vectors to their images in each linear transformation in \mathcal{Q} . While general quivers may have multiple arrows between vector spaces, companion matrices can only represent quivers which have a underlying graph with at most one linear transformation on each edge – this will not limit our study of type A_n quivers which satisfy this property.

For example a persistence quiver $P_4 = \cdot \leftarrow \cdot \leftarrow \cdot \leftarrow \cdot$ will have a companion matrix of the form

$$\begin{bmatrix} 0 & A_1 & & \\ & 0 & A_2 & \\ & & 0 & A_4 \\ & & & 0 \end{bmatrix}$$

whereas a zigzag quiver $Z_4 = \cdot \rightarrow \cdot \leftarrow \cdot \rightarrow \cdot$ will have a companion matrix of the form

$$\begin{bmatrix} 0 & & & \\ A_1 & 0 & A_2 & \\ & & 0 & \\ & & A_3 & 0 \end{bmatrix}$$

Quiver isomorphism classes are maintained by conjugation of the companion matrix by block-diagonal change of bases matrices. For example two persistence quivers of type $P_3 = \cdot \leftarrow \cdot \leftarrow \cdot$

with maps denoted by A and B respectively are isomorphic if there exists an invertible matrix $M = M_1 \oplus M_2 \oplus M_3$ acting on V such that

$$\begin{bmatrix} 0 & A_1 & & \\ & 0 & A_2 & \\ & & 0 & \end{bmatrix} = \begin{bmatrix} M_1 & & \\ & M_2 & \\ & & M_3 \end{bmatrix} \begin{bmatrix} 0 & B_1 & \\ & 0 & B_2 \\ & & 0 \end{bmatrix} \begin{bmatrix} M_1^{-1} & & \\ & M_2^{-1} & \\ & & M_3^{-1} \end{bmatrix}$$

An *indecomposable factorization* of the companion matrix A is a factorization $A = BTB^{-1}$, where B is an invertible (change of basis) matrix, and $T = \bigoplus_i I[b_i, d_i]$ is the matrix of indecomposables. We will allow for the indecomposable block to appear in any order but for exposition we will use the lexicographic partial order on \mathbb{Z}^2 to order the pairs (b_i, d_i) . For example, in the case where P_4 and Z_4 both have indecomposable matrices $T = I[1, 1] \oplus I[1, 4] \oplus [2, 3]$, the corresponding matrices are

$$T_{P_4} = \begin{bmatrix} 0 & & & & & \\ & 0 & 1 & 0 & 0 & \\ & 0 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 1 & \\ & 0 & 0 & 0 & 0 & \\ & & & & & 0 & 1 \\ & & & & & 0 & 0 \end{bmatrix} \quad T_{Z_4} = \begin{bmatrix} 0 & & & & & \\ & 0 & 0 & 0 & 0 & \\ & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & \\ & 0 & 0 & 1 & 0 & \\ & & & & & 0 & 1 \\ & & & & & 0 & 0 \end{bmatrix} \quad (3.2)$$

where the information for the indecomposable $I[1, 4]$ is colored in **red**. Note the indecomposable blocks all appear as adjacency matrices of sub-graphs of the underlying directed graph of the quiver. This means that even though the indecomposables are written with the same notation, the indecomposable matrices are not identical due to the different directions of arrows. The advantage of the indecomposable factorization is that it is easy to determine the lengths the indecomposables, but information about which vector spaces participate is obscured.

A *barcode factorization* of the companion matrix A is a factorization $A = B\Lambda B^{-1}$, where B is a *block-diagonal* invertible matrix (representing a quiver isomorphism), where the block sizes are compatible with dimensions of the quiver, and

$$\Lambda = PTP^T$$

is the barcode matrix, where P is a permutation that preserves the block structure of A . Alternatively, we'll say Λ is the *barcode form* of the quiver companion matrix A , or simply the barcode form of the quiver representation. Continuing the previous example, in the case where P_4 and Z_4 both

have barcode matrices $\Lambda \cong I[1, 1] \oplus I[1, 4] \oplus [2, 3]$, the corresponding matrix representations are

$$\Lambda_{P_4} = \begin{bmatrix} & 0 & 0 & & \\ & \textcolor{red}{1} & 0 & & \\ & & \textcolor{red}{1} & 0 & \\ & & 0 & 1 & \\ & & & & \textcolor{red}{1} \\ & & & & 0 \end{bmatrix} \quad \Lambda_{Z_4} = \begin{bmatrix} & & & & \\ 0 & \textcolor{red}{1} & & \textcolor{red}{1} & 0 \\ 0 & 0 & & 0 & 1 \\ & & & & \\ & & & \textcolor{red}{1} & 0 \end{bmatrix} \quad (3.3)$$

The information for the indecomposable $I[1, 4]$ is colored in **red**. Note that because the underlying graphs are different the matrices Λ are not equal even though the interval decomposition is superficially the same. In both quivers, the ranks of the vector spaces are 2, 2, 2, 1. The advantage of the barcode factorization is that B clearly represents a quiver isomorphism due to its block structure.

Extracting the intervals $I[a, b]$ from a barcode factorization requires tracing the image of maps through the quiver, and the advantage compared to the indecomposable decomposition is that the starting point of an interval is clear. We extract the intervals from Λ by sweeping through the blocks left-to-right

Algorithm 1 Barcode Extraction

```

1: Input: Barcode matrix  $\Lambda$ , ranks of vector spaces  $V_i$  and directions of arrows in quiver.
2: Result: Barcode  $\mathcal{B}$ 
3: for  $i = 1, \dots, n$  do
4:   for  $j = 1, \dots, \text{rank } V_i$  do
5:     if  $V_{i-1} \rightarrow V_i$  then
6:       if Row  $j$  in block  $i$  contains a non-zero in column  $j'$  of block  $i - 1$  then
7:         Extend the bar at index  $j'$  of block  $i - 1$  to have index  $j$  in block  $i$ .
8:       else
9:         Begin a bar with index  $j$  in block  $i$ 
10:      end if
11:    else if  $V_{i-1} \leftarrow V_i$  then
12:      if Column  $j$  in block  $i$  contains a non-zero in row  $j'$  of block  $i - 1$  then
13:        Extend the bar at index  $j'$  of block  $i - 1$  to have index  $j$  in block  $i$ .
14:      else
15:        Begin a bar with index  $j$  in block  $i$ 
16:      end if
17:    else
18:       $(i = 1)$ 
19:      Begin bar with index  $j$  in block 1
20:    end if
21:  end for
22: end for
23: return  $\mathcal{B}$ 

```

If we keep track of the indices used in each extension of a bar, we have the information necessary to form the permutation P so that $P\Lambda P^T = T$.

Definition 3.3.1.2. *A pivot matrix is a matrix in which every row and column has at most one non-zero element.*

Proposition 3.3.1.3. *A companion matrix is in barcode form if and only if its blocks are pivot matrices.*

Proof. If all blocks of a companion matrix are pivot matrices, then every iteration of the for-loop in Section 3.3.1 will find at most one index that can be used to extend a bar. The set of bars found by the algorithm gives the indecomposables, so the matrix is in barcode form.

If the matrix is in barcode form, we'll consider the representation of the map $A_i : V \rightarrow W$, where either $V = V_i, W = V_{i+1}$ or $V = V_{i+1}, W = V_i$. Because the matrix is in barcode form, every basis

vector $v \in V$ maps to either exactly one basis vector of W (continuing a bar), in which case the corresponding column of A_i has exactly one non-zero, or maps to zero (the bar ends at V), in which case the corresponding column of A_i is zero. Every basis vector $w \in W$ is either in the image of a basis vector of V , in which case the corresponding row of A_i has exactly one non-zero, or is the start of a new bar, in which case the row of A_i is zero. Thus A_i is a pivot matrix because it has at most one non-zero in each row and column. \square

3.3.2 Matrix passing algorithm

There is a correspondence between diagrams encoding quivers and block matrices encoding their companion matrices, and certain operations are easier to express using one notation or the other. In this section we establish some lemmas that apply to any quiver.

Lemma 3.3.2.1. *A change of basis (quiver isomorphism) at a single node via an invertible matrix M can be represented as*



Proof. This follows immediately from a change of basis on the central vector space in the diagram via Equation (3.1). \square

If the quiver is representable by a companion matrix, this diagrammatically encodes the quiver isomorphism

$$\begin{aligned}
 & \begin{bmatrix} & B_0 & \dots & B_m \\ A_0 & & & \\ \vdots & & & \\ A_n & & & \end{bmatrix} \\
 &= \begin{bmatrix} M & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix} \begin{bmatrix} & M^{-1}B_0 & \dots & M^{-1}B_m \\ A_0M & & & \\ \vdots & & & \\ A_nM & & & \end{bmatrix} \begin{bmatrix} M^{-1} & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix}
 \end{aligned}$$

We see that this only affects linear transformations that have the center node as a source or target. For any vector spaces that do not have an arrow to or from the center vector space are multiplied by an identity on both left and right and are unaffected.

Lemma 3.3.2.1 implies the following two corollaries which set forth the rules for our *matrix passing algorithms*.

Corollary 3.3.2.2. *Passing an invertible matrix M through a target yields*



Corollary 3.3.2.3. *Passing an invertible matrix M through a source yields*



Notice that we draw arrows from right to left in the above diagrams. This is simply because the matrix M is closest to the central node. If we write arrows right to left we would have the correct, but less natural looking example

$$\cdot \xrightarrow{B_0} \cdot \xrightarrow{\tilde{A}_0 M} \cdot \quad \cong \quad \cdot \xrightarrow{M B_0} \cdot \xrightarrow{\tilde{A}_0} \cdot$$

Generally, quivers on induced maps on homology can be derived from diagrams of chain maps.

The generic quiver computations are useful in representing a wide variety of computations. For example we can cast the problem of computing persistence homology from chain group information as a factorization of the following grid quiver. Since computing the barcodes entails a change of basis, it can be expressed using matrix passing algorithms using the generic quiver computation.

$$\begin{array}{ccccccc} \cdot & \xleftarrow{\partial_{1,0}} & \cdot & \xleftarrow{\partial_{1,1}} & \cdots & \xleftarrow{\partial_{1,n-2}} & \cdot & \xleftarrow{\partial_{1,n-1}} & \cdot \\ \downarrow \phi_{1,1} & & \downarrow \phi_{1,2} & & & & \downarrow \phi_{1,n-1} & & \downarrow \phi_{1,n} \\ \cdot & \xleftarrow{\partial_{2,0}} & \cdot & \xleftarrow{\partial_{2,1}} & \cdots & \xleftarrow{\partial_{2,n-2}} & \cdot & \xleftarrow{\partial_{2,n-1}} & \cdot \\ \downarrow \phi_{2,1} & & \downarrow \phi_{2,2} & & & & \downarrow \phi_{2,n-1} & & \downarrow \phi_{2,n} \\ \vdots & & \vdots & & \cdots & & \vdots & & \vdots \end{array}$$

As another example, here is the standard type A quiver equipped with some extra identity maps. When the factorization algorithm is applied to the type A sub-quiver, the identity maps will be modified and will record the basis change matrices required for the factorization. In this way we

see that these quiver diagrams and associated quiver computations are a useful tool for expressing these algorithms.

$$\begin{array}{ccccccc}
 \cdot & & \cdot & & \cdot & & \cdot \\
 \uparrow I & & \uparrow I & & \uparrow I & & \uparrow I \\
 \cdot & \xleftarrow{A_0} & \cdot & \xleftarrow{A_1} & \cdots & \xleftarrow{A_{n-2}} & \cdot & \xleftarrow{A_{n-1}} & \cdot \\
 \uparrow I & & \uparrow I & & \uparrow I & & \uparrow I \\
 \cdot & & \cdot & & \cdot & & \cdot
 \end{array}$$

$$\begin{array}{ccccccc}
 \cdot & & \cdot & & \cdot & & \cdot \\
 \uparrow B_0 & & \uparrow B_1 & & \uparrow B_{n-1} & & \uparrow B_n \\
 \cdot & \xleftarrow{E_0} & \cdot & \xleftarrow{E_1} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\
 \uparrow B_0^{-1} & & \uparrow B_1^{-1} & & \uparrow B_{n-1}^{-1} & & \uparrow B_n^{-1} \\
 \cdot & & \cdot & & \cdot & & \cdot
 \end{array}$$

Chapter 4

Quiver Algorithms

In this chapter, we will look at algorithms to compute the canonical form of type A quiver representations. As we saw in the previous chapter, we can reduce the computation of persistence barcodes and zigzag barcodes into computing a canonical form. Before we describe the algorithm, we need to introduce some linear algebra tools. We will then look at the algorithms for persistence and zigzag quivers. We will then look at a divide and conquer algorithm for zigzag quivers. Finally we will look at some numerical experiments.

4.1 Linear algebra tools

In this section we will introduce some linear algebra tools that we will use later in the algorithm. Let us start with some notations and definitions.

4.1.1 Notations and definitions

Triangular Matrices: Triangular matrices only contain non-zero elements on one side of a diagonal. Common triangular forms are lower and upper triangular matrices, where $L \in \mathbb{F}^{m \times n}$ is lower-triangular if $L_{i,j} = 0$ if $j > i$, and $U \in \mathbb{F}^{m \times n}$ is upper triangular if $U_{i,j} = 0$ if $j < i$. Triangular matrices are extremely useful because solving linear systems using backward substitution [6] is as expensive as regular matrix multiplication. That is, solving $Tx = b$ for unknown x is as expensive as forming the matrix-vector product Tx . Backward substitution can be used successfully even when a triangular matrix is not invertible, either because it is not square or because it is rank deficient, as long as b is in the column space of T (i.e. the linear system is consistent).

The J Matrix:

Definition 4.1.1.1. J is a square $n \times n$ matrix, such that

$$J_{ij} = \begin{cases} 1, & \text{if } i = n - j - 1 \\ 0, & \text{otherwise} \end{cases}$$

In other words, it is the anti-diagonal permutation matrix. Specifically when multiplied on the left, it reverses the row order. Similarly it reverses the column order when multiplied on the right. It is its own inverse

$$J^{-1} = J$$

A common operation is to conjugate with the J matrix, which reverses both row and column order and thus produces a reflection across the anti-diagonal. Note that this operation is distinct from taking the transpose of a matrix and cannot be expressed in terms of it.

$$JAJ^{-1} = JAJ = A'$$

The following are useful commutation relations between the J matrix and other matrix shapes

$$JL = UJ \quad \begin{array}{|c|c|} \hline \diagup & \diagdown \\ \hline \end{array} = \begin{array}{|c|c|} \hline \diagdown & \diagup \\ \hline \end{array} \quad (4.1)$$

$$JU = LJ \quad \begin{array}{|c|c|} \hline \diagdown & \diagup \\ \hline \end{array} = \begin{array}{|c|c|} \hline \diagup & \diagdown \\ \hline \end{array} \quad (4.2)$$

$$JE_L = \hat{E}_U J \quad \begin{array}{|c|c|} \hline \diagup & \diagdown \\ \hline \end{array} = \begin{array}{|c|c|} \hline \diagdown & \diagup \\ \hline \end{array} \quad (4.3)$$

$$JE_U = \hat{E}_L J \quad \begin{array}{|c|c|} \hline \diagdown & \diagup \\ \hline \end{array} = \begin{array}{|c|c|} \hline \diagup & \diagdown \\ \hline \end{array} \quad (4.4)$$

Pivot Matrices:

Definition 4.1.1.2. A pivot matrix is a matrix in which every row and column has at most one non-zero element.

In the context of matrix factorizations used in this dissertation, the non-zero element will always be 1 (the multiplicative identity of the field \mathbb{F}) by convention. The term pivot matrix refers to its use in recording pivots (last nonzeros of rows or columns) when computing matrix factorizations. Pivot matrices are similar to permutation matrices in the sense that they map a single basis element to a single basis element, but contain the possibility that some rows and columns can be entirely zero. Thus they are not generally invertible.

Lemma 4.1.1.3. The class of pivot matrices is closed under multiplication

Proof. For a pivot matrix Q , define $i(j)$ to be the index of the non-zero row of column j if column j has a pivot, and $i(j) = \infty$ otherwise. Additionally, define $i(\infty) = \infty$. Thus, we can write columns

of Q as

$$Q[j] = e_{i(j)}$$

where $e_\infty = 0$.

Let Q_1 and Q_2 be pivot matrices with compatible dimensions to form the product $A = Q_2 Q_1$, then column j of A can be written as

$$Ae_j = Q_2(Q_1 e_j) = Q_2 e_{i_1(j)} = e_{i_2(i_1(j))}$$

So column $A[j]$ has at most one nonzero, with pivot $i_2(i_1(j))$. \square

Echelon Pivot Matrices: Echelon pivot matrices are pivot matrices with added structure. There are 4 types we consider

$$E_L = \begin{bmatrix} \diagdown \\ \square \end{bmatrix} \quad E_U = \begin{bmatrix} \square \\ \diagup \end{bmatrix} \quad (4.5)$$

$$\hat{E}_L = \begin{bmatrix} \diagup \\ \square \end{bmatrix} \quad \hat{E}_U = \begin{bmatrix} \square \\ \diagdown \end{bmatrix} \quad (4.6)$$

E_L and \hat{E}_U contain the pivots for variants of the column echelon form of a matrix, and E_U and \hat{E}_L contain the pivots for variants of the row echelon form of a matrix. The L and U subscript indicates whether the matrix is lower or upper triangular.

Definition 4.1.1.4. A matrix has the E_L shape if it is the sum of rank 1 matrices created from basis vectors

$$E_L = \sum_{(i,j) \in S} e_i e_j^T$$

The set $S \subset \{1, \dots, m\} \times \{1, \dots, n\}$ contains the locations of the pivots. Since E_L is a pivot matrix, for every j , there must be a unique i , therefore the pairs can be written as $(i(j), j)$. The function $i(j)$ is defined on the subset of columns that have a pivot, and must satisfy the following properties

1. $j_1 < j_2 \implies i(j_1) < i(j_2)$ on the domain of $i(j)$
2. For every j_1, j_2 s.t. $j_1 < j_2$ and $A[j_1] = 0 \implies A[j_2] = 0$

The other echelon pivot matrices can be defined in terms of the E_L shape.

Definition 4.1.1.5. A matrix is of shape E_U if its transpose is of shape E_L

$$(E_U)^T = E_L$$

Definition 4.1.1.6. A matrix is of shape \hat{E}_L if its J -Conjugate is of shape E_L

$$J \hat{E}_L J = E_L$$












Symbol	Meaning	Shape
A	Arbitrary matrix	
D	Diagonal matrix	
L	Lower-triangular	
U	Upper-triangular	
T	Any triangular form	
S	Schur Complement	
P	Permutations	
I	Identity	
J	Anti-diagonal permutation	
E	Echeleon-diagonal	
E_L	Echelon pivot lower	
\hat{E}_L	Echelon pivot lower	
E_U	Echelon pivot upper	
\hat{E}_U	Echelon pivot upper	
Q	Pivot matrix	

Figure 4.1: Notation for different matrices, along with pictorial symbols

Definition 4.1.1.7. *A matrix is of shape \hat{E}_U if its J-Conjugate is of shape E_U*

$$J\hat{E}_U J = E_U$$

4.1.2 Triangular Factorizations

In this section, we will discuss computing decompositions of a matrix A into triangular matrices with row or column pivoting. Specifically we will start with the LEUP decomposition. Variants include PLEU, UELP and PUEL form, which can be derived using the LEUP decomposition of either transposed or J-Conjugated versions of A . These factorizations are all variants of the standard LU decomposition [6], but we will need these specific forms for our quiver algorithm.

Given a matrix A , we will describe an algorithm that will maintain a block invariant at each step i

$$A = \begin{bmatrix} L_{11} & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} E_{11} & \\ & \tilde{A} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ & I \end{bmatrix} P \quad (4.7)$$

Algorithm 2 LEUP factorization

```

1: Input:  $m \times n$  matrix  $A$ 
2: Result: Factorization  $A = LEUP$ 
3:  $L = I_m$ 
4:  $U = I_n$ 
5:  $P = I_n$ 
6:  $i = 1, j = 1$ 
7: while  $i \leq m$  &  $j \leq n$  do
8:   if row  $i$  has a non-zero in column  $j' \geq j$  then
9:     Swap columns  $j, j'$  in  $A$ 
10:    Take Schur complement with respect to  $i, j$  entry
11:     $i = i + 1$ 
12:     $j = j + 1$ 
13:  else
14:     $i = i + 1$ 
15:  end if
16: end while

```

Proof of Correctness: We will show that the invariant Equation (4.7) is maintained at each step of the algorithm. Note that the loop increments i each iteration, so we use i as our index. For rows of A , as well as rows and columns of L , we will use the block index set $1 = \{0, \dots, i-1\}$, and the block index set $2 = \{i+1, \dots, m-1\}$, and for columns of A , as well as rows and columns of U we will use the block index set $1 = \{0, \dots, j-1\}$ and block index set $2 = \{j+1, \dots, n-1\}$.

Assume the invariant is maintained at the beginning of iteration i . We can break up the matrices into

$$A = \begin{bmatrix} L_{11} & & \\ L_{i1} & 1 & \\ L_{21} & & I \end{bmatrix} \begin{bmatrix} E_{11} & & \\ & A_{ij} & A_{i2} \\ & A_{2j} & A_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{1j} & U_{22} \\ & 1 & \\ & & I \end{bmatrix} P$$

In the case that there are no non-zero entries in A_{ij} or A_{i2} , we simply increment i (in the else clause of the while loop), and the invariant is trivially maintained.

In the case there is a non-zero entry in A_{ij} or A_{i2} , assume we have already permuted it to the A_{ij} position, and used the relation ?? to move the permutation to the right. We can write the interior matrix as

$$\begin{bmatrix} E_{11} & & \\ & A_{ij} & A_{i2} \\ & A_{2j} & A_{22} \end{bmatrix} = \begin{bmatrix} I & & \\ & I & \\ & A_{2j}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} E_{11} & & \\ & A_{ij} & \\ & & S \end{bmatrix} \begin{bmatrix} I & & \\ & I & A_{11}^{-1}A_{12} \\ & & I \end{bmatrix}$$

where S is the Schur complement $S = A_{22} - A_{2j}A_{ij}^{-1}A_{i2}$. We can then pass off the left matrix to L and the right matrix to U , and now group A_{ij} with the echelon block E_{11} .

Note that because we may increment i without incrementing j , that the matrix E will be of type E_L .

Other triangular factorizations

$$A = LE_LUP \quad \text{[grey] = [orange triangle] [blue diagonal] [green triangle] [purple dots]} \quad (4.8)$$

$$A = PLE_UU \quad \text{[grey] = [purple dots] [orange triangle] [blue diagonal] [green triangle]} \quad (4.9)$$

$$A = U\hat{E}_ULP \quad \text{[grey] = [green triangle] [blue diagonal] [orange triangle] [purple dots]} \quad (4.10)$$

$$A = PU\hat{E}_LL \quad \text{[grey] = [purple dots] [green triangle] [blue diagonal] [orange triangle]} \quad (4.11)$$

All of the above factorizations can be shown to be equivalent to the LE_LUP factorization, using transposes and conjugation with J matrices.

For example consider the case,

$$A = (A^T)^T = (LE_LUP)^T = P^T U^T (E_L)^T L^T = \tilde{P} \tilde{L} E_U \tilde{U}$$

Thus the PLE_UU factorization is equivalent to the LE_LUP factorization of the transpose. Similarly the $U\hat{E}_ULP$ can be expressed using J conjugation,

$$A = JJAJJ = J(JAJ)J = J\hat{A}J$$

Now we can replace \hat{A} with its LE_LUP factorization, and apply the commutation relations of J .

$$J\hat{A}J = JLE_LUPJ = \hat{U}JE_LUPJ = \hat{U}\hat{E}_UJUPJ = \hat{U}\hat{E}_U\hat{L}JPJ = \hat{U}\hat{E}_U\hat{L}\hat{P}$$

Thus we have found the $U\hat{E}_ULP$ factorization of A .

If we apply the PLE_UU factorization for \hat{A} instead, we get,

$$J\hat{A}J = JPLE_UUJ = \hat{P}JLE_UUJ = \hat{P}\hat{U}JE_UUJ = \hat{P}\hat{U}\hat{E}_LJUJ = \hat{P}\hat{U}\hat{E}_L\hat{L}$$

This gives us the $PU\hat{E}_LL$ factorization for the matrix A .

LQU Factorization

The LQU factorization is different from the triangular factorizations introduced before. It does not perform any pivoting and therefore there is no auxiliary permutation matrix produced. Instead, we sacrifice structure in the pivot matrix and obtain a general pivot matrix Q as opposed to echelon pivot matrices.

Algorithm 3 LQU factorization

```

1: Input:  $m \times n$  matrix  $A$ 
2: Result: Factorization  $A = LQU$ 
3:  $L = I_m$ 
4:  $Q = A$ 
5:  $U = I_n$ 
6:  $j = 1$ 
7: while  $j \leq n$  do
8:   if column  $j$  has a non-zero in a non-pivot row, let the first such row be  $i$  then
9:     Zero out all elements in non-pivot rows in column  $j$  below  $i$ 
10:    Record row operations in  $L$ 
11:    Mark  $i$  as pivot row
12:     $j = j + 1$ 
13:   else
14:      $j = j + 1$ 
15:   end if
16: end while
17:  $i = 1$ 
18: while  $i \leq m$  do
19:   if row  $i$  is a pivot-row with pivot at  $j$  then
20:     Zero out all elements after  $j$ 
21:     Record column operations in  $U$ 
22:      $i = i + 1$ 
23:   else
24:      $i = i + 1$ 
25:   end if
26: end while

```

Proof of Correctness: In order to see that the above algorithm is correct, we first note that both the row operations and column operations are triangular, i.e. row i is used to eliminate rows at positions greater than i and column j is used to eliminate columns at positions greater than j . Thus the recorded L and U matrices are indeed lower and upper triangular respectively.

Now it is left to prove that the resultant matrix Q , has pivot structure. If we prove that the only non-zeros at the end of the algorithm are the pivots then we are done, as pivots are chosen such that no two of them share a row or column. Let us prove this by contradiction, suppose there is a non-zero that was not eliminated at the end of the algorithm. It has to be either in a non-pivot row or its column position is before a pivot in a pivot row, otherwise it would have been eliminated by the column operations. Such an element cannot exist as it should have been eliminated by row

operations by a pivot above it in its column. The pivot cannot be below as it would imply that we did not pick the first non-zero in a non-pivot row when choosing the pivot for this column.

4.1.3 Shape Commutation relations

Now, we'll consider shape commutation relations of echelon matrices with triangular matrices. We will first show the following commutation relationship, and derive others from this.

Proposition 4.1.3.1. *Given an echelon pivot matrix E_L and lower triangular matrix L , we can rewrite the product $E_L L$ in the following way*

$$E_L L = \tilde{L} E_L$$

Where \tilde{L} matrix is also a lower triangular matrix.

Proof. In terms of matrix shapes this commutation looks like the following

$$\begin{array}{|c|} \hline \text{blue diagonal} \\ \hline \end{array} \begin{array}{|c|} \hline \text{orange triangle} \\ \hline \end{array} = \begin{array}{|c|} \hline \text{orange triangle} \\ \hline \end{array} \begin{array}{|c|} \hline \text{blue diagonal} \\ \hline \end{array}$$

Note that this is just the commutation of the shapes, In general L and \tilde{L} are not the same matrices, or even the same dimensions. We will first characterize left and right multiplication by the E_L matrix.

Consider the product of E_L with an arbitrary matrix A

$$(E_L A)_{kl} = \sum_p (E_L)_{kp} A_{pl}$$

From the definition of the shape of E_L , we know that its entries are 1 only when $p = j(k)$, thus we get

$$(E_L A)_{kl} = \sum_p (E_L)_{kp} A_{pl} = (E_L)_{k, j(k)} A_{j(k), l} = A_{j(k), l}$$

Here we use the convention that if a particular k, l index pair is not assigned any value, it is by default 0. Similarly, if we are attempting to apply the function $j(\cdot)$ on an index not in its domain, the appropriate matrix element is 0. Multiplication on the right follows similarly

$$(A E_L)_{kl} = \sum_p A_{kp} (E_L)_{pl} = A_{k, i(l)} (E_L)_{i(l), l} = A_{k, i(l)}$$

We will now show that \tilde{L} , constructed in the following way will satisfy the proposition.

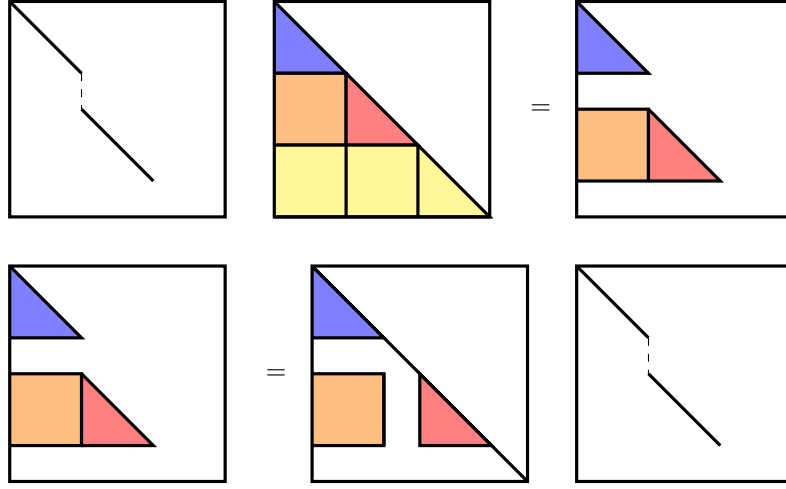


Figure 4.2: Pictorial representation of the shape commutation relationship in Proposition 4.1.3.1

$$\tilde{L}_{kl} = \begin{cases} L_{j(k),j(l)}, & \text{if } k \in \text{Domain}(j(\cdot)) \text{ and } l \in \text{Domain}(j(\cdot)) \\ 1, & \text{if } k = l \text{ and } k \notin \text{Domain}(j(\cdot)) \\ 0, & \text{otherwise} \end{cases}$$

Note that the above construction is not unique, we have opted to set the diagonal of otherwise zero columns to 1, this makes the matrix invertible, which will be useful later on. Next we will show that this construction is indeed lower triangular.

Suppose $k < l$, then we have from the definition of the shape of E_L that $j(k) < j(l)$ which implies that $L_{j(k),j(l)} = 0$, as L is a lower triangular matrix. This shows that for $k < l$, $\tilde{L}_{kl} = 0$, hence \tilde{L} is lower triangular. Now we will compute $E_L L$ and $\tilde{L} E_L$ and show that they are equal.

$$(E_L L)_{kl} = L_{j(k),l}$$

$$(\tilde{L} E_L)_{kl} = \tilde{L}_{k,i(l)} = L_{j(k),j(i(l))} = L_{j(k),l}$$

Since $i(\cdot)$ is the inverse function of $j(\cdot)$, $i(l)$ will always be in the domain of $j(\cdot)$, thus

$$E_L L = \tilde{L} E_L$$

□

Proposition 4.1.3.2. *The following other shape commutation relations also hold*

$$L\hat{E}_L = \hat{E}_L\tilde{L} \quad \begin{array}{|c|c|} \hline \text{orange square} & \text{blue square} \\ \hline \end{array} = \begin{array}{|c|c|} \hline \text{blue square} & \text{orange square} \\ \hline \end{array} \quad (4.12)$$

$$UE_U = E_U\tilde{U} \quad \begin{array}{|c|c|} \hline \text{green square} & \text{blue square} \\ \hline \end{array} = \begin{array}{|c|c|} \hline \text{blue square} & \text{green square} \\ \hline \end{array} \quad (4.13)$$

$$\hat{E}_U U = \tilde{U} \hat{E}_U \quad \begin{array}{|c|c|} \hline \text{blue square} & \text{green square} \\ \hline \end{array} = \begin{array}{|c|c|} \hline \text{green square} & \text{blue square} \\ \hline \end{array} \quad (4.14)$$

Proof. Taking transpose on the commutation result for E_L

$$(E_L L)^T = (\tilde{L} E_L)^T$$

$$L^T E_L^T = E_L^T \tilde{L}^T$$

Rewriting to denote the shapes we get,

$$UE_U = E_U\tilde{U}$$

Taking the J-Conjugate of the E_L commutation result we have,

$$J(E_L L)J = J(\tilde{L} E_L)J$$

$$JE_L L J = \hat{E}_U J L J = \hat{E}_U U$$

$$J\tilde{L} E_L J = \tilde{U} J E_L J = \tilde{U} \hat{E}_U$$

We get the commutation result for \hat{E}_U

$$\hat{E}_U U = \tilde{U} \hat{E}_U$$

Taking the transpose of the above result, we get,

$$(\hat{E}_U U)^T = (\tilde{U} \hat{E}_U)^T$$

$$U^T \hat{E}_U^T = \hat{E}_U^T \tilde{U}^T$$

Rewriting to denote shapes,

$$L\hat{E}_L = \hat{E}_L\tilde{L}$$

□

4.2 Algorithm for Persistent type quivers

At a high level, the algorithm will put each matrix in pivot matrix form. This is accomplished in two passes - we will work from left to right on the first pass, and then right to left on the second pass. Note that we could equally work in the opposite order (see fig. 4.4). We will use diagrams to notate the steps of the algorithm, keeping in mind that we can keep track of the invertible basis transformation for each of the steps. A pictorial description is contained in Section 4.2. We will first apply the LE_LUP factorization to A_0

$$\begin{array}{ccccccc} \cdot & \xleftarrow{A_0} & \cdot & \xleftarrow{A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0 E_0 U_0 P_0} & \cdot & \xleftarrow{A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \end{array}$$

We can now use matrix passing to move both U_0 and P_0 matrices, as they are both invertible. We then multiply the matrices to get $\tilde{A}_1 = U_0 P_0 A_1$

$$\begin{array}{ccccccc} \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{U_0 P_0 A_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{\tilde{A}_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \end{array}$$

We can now apply this procedure to \tilde{A}_1 and then iterate through the rest of the maps in the quiver representation.

$$\begin{array}{ccccccc} \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{L_1 E_1 U_1 P_1} & \cdots & \xleftarrow{A_{n-1}} & \cdot \\ & & \vdots & & & & \\ \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{L_1 E_1} & \cdots & \xleftarrow{L_{n-1} E_{n-1} U_{n-1} P_{n-1}} & \cdot \end{array}$$

Applying matrix passing on the final edge, we can remove the factor $U_{n-1} P_{n-1}$

$$\begin{array}{ccccccc} \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{L_1 E_1} & \cdots & \xleftarrow{L_{n-1} E_{n-1} U_{n-1} P_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0 E_0} & \cdot & \xleftarrow{L_1 E_1} & \cdots & \xleftarrow{L_{n-1} E_{n-1}} & \cdot \end{array}$$

Next we can initiate the leftward pass by moving the lower triangular matrices leftward using the shape commutation relations at each step. We do so, as follows,

$$\begin{array}{ccccccc} \cdot & \xleftarrow{L_0 E_0} & \cdots & \xleftarrow{L_{n-2} E_{n-2}} & \cdot & \xleftarrow{L_{n-1} E_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0 E_0} & \cdots & \xleftarrow{L_{n-2} E_{n-2} L_{n-1}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\ \cdot & \xleftarrow{L_0 E_0} & \cdots & \xleftarrow{\tilde{L}_{n-2} E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \end{array}$$

Here we have used the following shape commutation relation

$$L_{n-2} E_{n-2} L_{n-1} = \tilde{L}_{n-2} E_{n-2}$$

Applying iteratively over the rest of the edges in a right-to-left sweep, we obtain

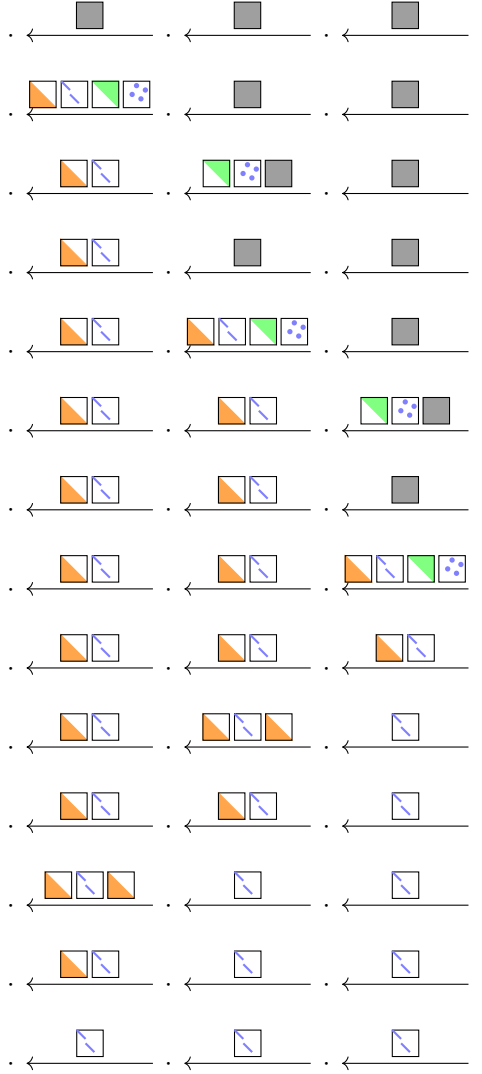
$$\begin{array}{ccccccc}
\cdot & \xleftarrow{L_0 E_0} & \cdots & \xleftarrow{\tilde{L}_{n-2} E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\
& & & \vdots & & \vdots & \\
& & & \tilde{L}_0 E_0 & \cdots & \xleftarrow{E_{n-2}} & \cdot \xleftarrow{E_{n-1}} \cdot
\end{array}$$

Finally, we can remove the last factor \tilde{L}_0 , by matrix passing

$$\begin{array}{ccccccc}
\cdot & \xleftarrow{\tilde{L}_0 E_0} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot \\
\cdot & \xleftarrow{E_0} & \cdots & \xleftarrow{E_{n-2}} & \cdot & \xleftarrow{E_{n-1}} & \cdot
\end{array}$$

We have reduced all the matrices to pivot matrix form.

A pictorial representation of the above algorithm is depicted below.



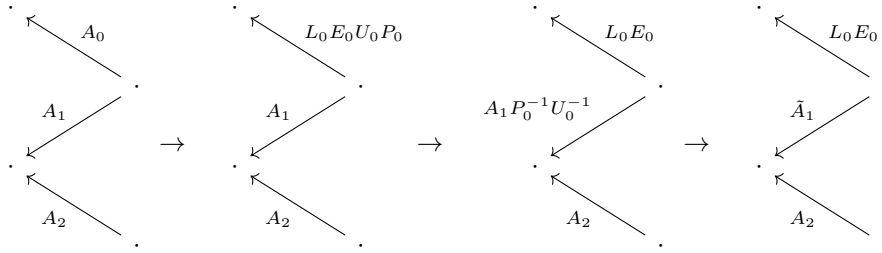
4.3 Algorithm for Zigzag quivers

In the general A-type Quiver diagram, the arrows can be in either direction. So far we have seen an algorithm that works when all the arrows are in the same direction.

To illustrate how the algorithm would work with arbitrary arrow directions, consider the following zigzag quiver. We have applied the first few steps of the algorithm to it.

$$\begin{array}{ccccccc}
 \cdot & \xleftarrow{A_0} & \cdot & \xrightarrow{A_1} & \cdot & \xleftarrow{A_2} & \cdot \\
 \cdot & \xleftarrow{L_0 E_0 U_0 P_0} & \cdot & \xrightarrow{A_1} & \cdot & \xleftarrow{A_2} & \cdot \\
 \cdot & \xleftarrow{L_0 E_0} & \cdot & \xrightarrow{A_1 P_0^{-1} U_0^{-1}} & \cdot & \xleftarrow{A_2} & \cdot \\
 \cdot & \xleftarrow{L_0 E_0} & \cdot & \xrightarrow{\tilde{A}_1} & \cdot & \xleftarrow{A_2} & \cdot
 \end{array}$$

The above transformation might be easier to see if the diagram was in the following form,

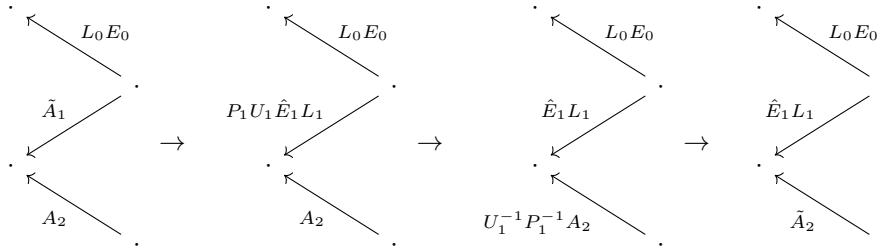


Here we have redrawn the quiver so that all the arrows are going from right to left, this is so that that matrix passing rules are more intuitive to apply.

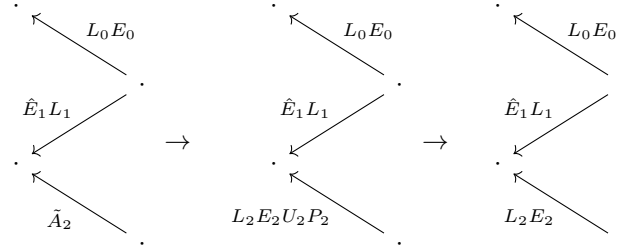
For the next step, as the arrow is reversed we cannot use the LE_LUP factorization. This would result in matrices that cannot be commuted during the second sweep. To handle this case, we use the $PU\hat{E}_LL$ factorization instead.

$$\tilde{A}_1 = P_1 U_1 \hat{E}_1 L_1$$

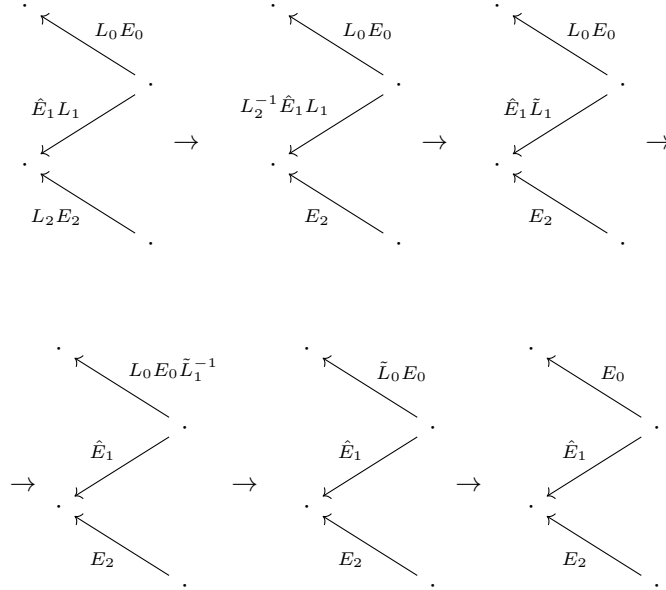
Applying matrix passing to move the factored matrices to the next edge, we get



For the last edge, we can again apply the LE_LUP factorization,



Now we can perform the reverse sweep as follows,



During the reverse sweep we used the following commutation relations

$$L_2^{-1} \hat{E}_1 L_1 = \hat{E}_1 \tilde{L}_1$$

$$L_0 E_0 \tilde{L}_1^{-1} = \tilde{L}_0 E_0$$

Thus we have reduced all matrices to pivot matrices, the barcodes can be directly read of from them.

Here we had to use a different factorization depending on the arrow direction, we can now use this to generalize the algorithm to a type-A quiver with arbitrary arrow directions.

4.3.1 General Sequential Quiver Algorithm

For arbitrary directions of the arrows, as long as we use the correct factorization for each of the arrow directions, we can use the shape commutation relations in the reverse sweep to reduce all the matrices to echelon pivot form.

Algorithm 4 Obtain Barcode factorization of type A quiver: Rightward-initial

```

1: Input: Type  $A$  quiver.
2: Result: Barcode form of quiver.
3: for forward pass do
4:   if  $\leftarrow$  then
5:     Apply  $LE_LUP$  Factorization
6:     Matrix pass  $UP$  factors
7:   else
8:     Apply  $PU\hat{E}_LL$  Factorization
9:     Matrix pass  $PU$  factors
10:  end if
11: end for
12: Matrix pass  $L$  factor on the last edge
13: for backward pass do
14:   if  $\leftarrow$  then
15:     Commute  $L_1E_LL_2 = \tilde{L}_1E_L$ 
16:   else
17:     Commute  $L_1\hat{E}_LL_2 = \hat{E}_L\tilde{L}_2$ 
18:   end if
19: end for
20: Matrix pass the remaining  $L$  factor on the first edge.
```

We can also initiate the first sweep from the right to the left, to get a leftward initial algorithm. All the examples shown above initiate the first sweep from the left. For a general initial sweep direction and arrow direction, The tables fig. 4.3 and fig. 4.4 specify the factorization and commutation relation to use.

	Rightward Initial	Leftward Initial
\leftarrow	LE_LUP	PLE_UU
\rightarrow	$PU\hat{E}_LL$	$U\hat{E}_ULP$

Figure 4.3: The factorization to use for the first sweep.

We note that the commutation relations established earlier do not change the nonzero structure of the E matrices. Thus, it is possible to extract the barcode without performing the backward pass

	Rightward Initial	Leftward Initial
\leftarrow	$U_1 E_U U_2 = E_U \tilde{U}_2$	$L_1 E_L L_2 = \tilde{L}_1 E_L$
\rightarrow	$U_1 \hat{E}_U U_2 = \tilde{U}_1 \hat{E}_U$	$L_1 \hat{E}_L L_2 = \hat{E}_L \tilde{L}_2$

Figure 4.4: The commutation to use for the second sweep.

of Algorithm 4 if one does not care to form the change of basis.

4.4 Divide and Conquer parallel algorithm for zigzag quivers

In this section, we will show how we can divide a type-A quiver into two parts and perform the computation in parallel for each of the parts. The results can then be combined to give the full barcode factorization of the entire quiver. Before we get into the details of the algorithm, we will first introduce a technique to transform pivot matrices from one type to another.

E Matrix transformations

In this section we will see how we can factorize any pivot matrix Q into a permutation and an echelon pivot matrix

Proposition 4.4.0.1. *Given any pivot matrix Q , we can rewrite it as the following*

$$Q = E_L P \tag{4.15}$$

$$Q = P E_U \tag{4.16}$$

$$Q = \hat{E}_U P \tag{4.17}$$

$$Q = P \hat{E}_L \tag{4.18}$$

$$\tag{4.19}$$

where P is an appropriate permutation matrix.

Proof. We apply the $LE_L UP$, $PLE_U U$, $U\hat{E}_U LP$ and $PU\hat{E}_L L$ factorizations to Q and note that the triangular matrices are just the identity matrices. This is because the triangular matrices are produced to eliminate one entry with another entry in the same row or column, but such a situation cannot occur in a pivot matrix Q , so only permutation operations are performed in the factorization, resulting in a permutation matrix and an echelon pivot matrix. \square

Divide and conquer

Consider a quiver Q_γ with general arrow directions,

$$\cdot \xleftarrow{A_0} \cdot \xleftarrow{A_1} \cdots \xleftarrow{A_{n-1}} \cdot$$

We will divide it into two parts at position m , to give us two sub-quivers Q_α and Q_β

$$\cdot \xleftarrow{A_0} \cdots \xleftarrow{A_{m-1}} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot$$

Next we will introduce three auxiliary edges containing identity maps to aide us in the computations by acting as a place holder for matrices.

$$\cdot \xleftarrow{I} \cdot \xleftarrow{A_0} \cdots \xleftarrow{A_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot \xleftarrow{I} \cdot$$

We can now perform two versions of the sequential algorithm in parallel. For quiver Q_α , we will use the rightward-initial algorithm (Algorithm 4). Notice that the terminal matrices are collected in the auxiliary edges, they will be important when we merge the results of the two sub-quivers.

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{U_\alpha P_\alpha} \cdot \xleftarrow{A_m} \cdots \xleftarrow{A_{n-1}} \cdot \xleftarrow{I} \cdot$$

For the quiver Q_β , we will use the leftward-initial sequential algorithm. This allows us to collect both the permutation matrices in the center auxiliary edge.

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{U_\alpha P_\alpha P_\beta L_\beta} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

We can now multiply out the matrices in the center auxiliary edge and perform an LQU factorization using Algorithm 3

$$U_\alpha P_\alpha P_\beta L_\beta = C_\gamma$$

Since all the matrices $U_\alpha, P_\alpha, P_\beta, L_\beta$ are invertible, the pivot matrix Q_γ , will be full rank, and thus turn out to be a permutation matrix P_γ

$$C_\gamma = L_\gamma Q_\gamma U_\gamma = L_\gamma P_\gamma U_\gamma$$

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{L_\gamma P_\gamma U_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

The matrices E_0 to E_{m-1} were produced by the rightward-initial algorithm, so they are of shape E_L or \hat{E}_L depending on the arrow directions. They can be used to commute the L_γ factor all the way to the left. This process is very similar to the second sweep of the rightward-initial algorithm. Similarly, the matrices E_m to E_{n-1} are of shape E_U or \hat{E}_U . These matrices can be used to commute the U_γ factor towards the right in a manner similar to the second sweep of the leftward-initial algorithm.

$$\cdot \xleftarrow{L_\alpha \hat{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{P_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\hat{U}_\gamma U_\beta} \cdot$$

These propagated factors can now be multiplied and we get one lower triangular factor \tilde{L}_γ on the left auxiliary edge and one upper triangular factor \tilde{U}_γ on the right auxiliary edge leaving a permutation matrix P_γ in the center auxiliary edge. We can think of the result as the "LQU" factorization of the quiver Q_γ

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{P_\gamma} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot$$

At this stage, we technically have a valid barcode factorization and if this is the entire quiver we could stop here, but since we want to apply this algorithm recursively, we want to convert this into the result of either a leftward-initial or rightward-initial algorithm. This can be done by propagating the permutation matrix P_γ either leftwards or rightwards using the E matrix transformations discussed in Section 4.4. If we wish to obtain the result of the rightward-initial algorithm, then we propagate right. Note this transforms the E_U and \hat{E}_U matrices in Q_β to E_L and \hat{E}_L . We will denote the transformed matrices by \tilde{E}_m to \tilde{E}_{n-1} .

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{\tilde{E}_m} \cdots \xleftarrow{\tilde{E}_{n-1}} \cdot \xleftarrow{\tilde{P}_\gamma \tilde{U}_\gamma} \cdot$$

We can now drop the auxiliary identity map in the center, to obtain the rightward-initial merge.

$$\cdot \xleftarrow{\tilde{L}_\gamma} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{\tilde{E}_m} \cdots \xleftarrow{\tilde{E}_{n-1}} \cdot \xleftarrow{\tilde{P}_\gamma \tilde{U}_\gamma} \cdot$$

Alternatively, we can also propagate the permutation matrix leftwards, to obtain the result of the leftward initial algorithm. This would transform the E_L and \hat{E}_L matrices into E_U and \hat{E}_U in Q_α

$$\begin{aligned} &\cdot \xleftarrow{\tilde{L}_\gamma \tilde{P}_\gamma} \cdot \xleftarrow{\tilde{E}_0} \cdots \xleftarrow{\tilde{E}_{m-1}} \cdot \xleftarrow{I} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot \\ &\cdot \xleftarrow{\tilde{L}_\gamma \tilde{P}_\gamma} \cdot \xleftarrow{\tilde{E}_0} \cdots \xleftarrow{\tilde{E}_{m-1}} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{\tilde{U}_\gamma} \cdot \end{aligned}$$

Note that the results are not exactly equal to the result you would obtain from applying either a rightward-initial or leftward-initial algorithm on the entire quiver. While the echelon matrices are of the right shape, the terminal matrices $\tilde{P}_\gamma \tilde{U}_\gamma$ or $\tilde{L}_\gamma \tilde{P}_\gamma$ appear in reversed order. However this does not matter if this is used to merge barcode factorizations at a higher level. At a higher level of the recursion, we would have the following product in the center auxiliary edge,

$$\cdot \xleftarrow{L_\alpha} \cdot \xleftarrow{E_0} \cdots \xleftarrow{E_{m-1}} \cdot \xleftarrow{P_\alpha U_\alpha L_\beta P_\beta} \cdot \xleftarrow{E_m} \cdots \xleftarrow{E_{n-1}} \cdot \xleftarrow{U_\beta} \cdot$$

They can still be multiplied out and the LQU factorization can be performed without affecting the rest of the algorithm.

$$P_\alpha U_\alpha L_\beta P_\beta = C_\gamma$$

$$C_\gamma = L_\gamma P_\gamma U_\gamma$$

Thus we have seen how a divide and conquer algorithm can be used to compute the barcode factorization of a quiver in two parts. We can now apply this recursively until the size of the quiver is small enough that it is more efficient to apply one of the sequential algorithms.

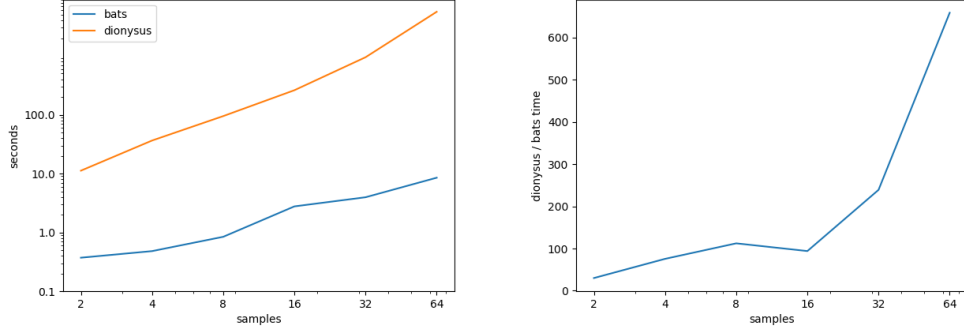


Figure 4.5: Time to compute zigzag homology of a zigzag diagram computed on subsamples of a noisy circle. Normal noise with variance 0.1 is added to points sampled from a unit circle. 200 points are contained in each subsample, and $r = 0.35$. The horizontal axis indicates the number of samples used in the diagram. Left: time to compute zigzag homology for both BATS and Dionysus. Right: Speedup seen using BATS instead of Dionysus. At the right hand side, BATS is over 600x faster.

4.5 Numerical Examples

We'll now investigate several experiments which demonstrate the flexibility of our methodology. Several preliminary timing results are presented based on computations done on an Intel i7-6820HQ CPU capable of running 8 threads concurrently.

4.5.1 Subsets and Rips Complexes

Let $\mathbf{X}_0, \mathbf{X}_1, \dots \subseteq X$ be finite subsets of a space, and $r_0, r_1, \dots \in \mathbb{R}$ be parameters for the Rips construction. We can construct a zigzag through unions of subsets

$$\mathcal{R}(\mathbf{X}_0; r_0) \hookrightarrow \mathcal{R}(\mathbf{X}_0 \cup \mathbf{X}_1, \max\{r_0, r_1\}) \hookleftarrow \mathcal{R}(\mathbf{X}_1; r_1) \hookrightarrow \dots \quad (4.20)$$

The case where $r = r_0 = r_1 = \dots$ was proposed in [3] as a “topological bootstrapping” method to investigate the robustness of homological features for samples \mathbf{X}_i sampled uniformly from some larger point cloud \mathbf{X} , and initial investigations were performed by Tausz [12]. A single value of r is also used in the case of [4] where the \mathbf{X}_i come from thickened level sets of a map $f : \mathbf{X} \rightarrow \mathbb{R}$

$$\mathbf{X}_i = f^{-1}((a_i, b_i))$$

where $(a_i, b_i) \cap (a_{i+1}, b_{i+1}) \neq \emptyset$.

As seen in fig. 4.5, the implementation in BATS displays a large performance advantage over Dionysus for subsampled Rips constructions, especially as the length of the diagram grows.

Another application of a zigzag of this form is to approximate the persistent homology of the full

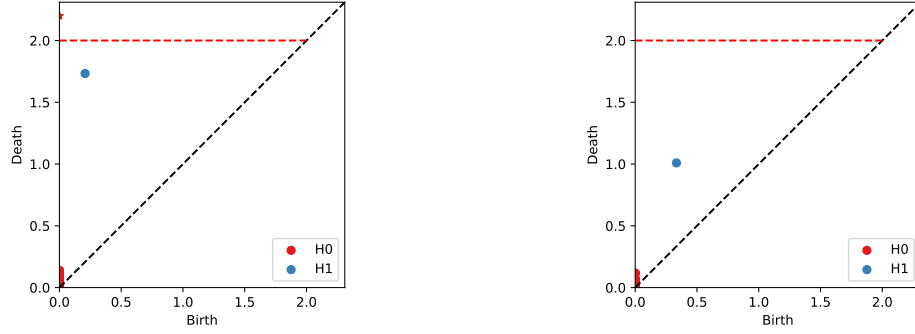


Figure 4.6: Left: persistence diagram for Rips filtration on 200 points sampled from the unit circle. The unoptimized reduction algorithm runs in 20 seconds in BATS. Right: An approximate persistence diagram created using the discrete Morozov zigzag construction in [11] using the suggested parameters. The zigzag computation takes approximately 0.5 seconds in BATS. While the birth and death times are not identical, both diagrams qualitatively display the same information, namely a single connected component and a robust H_1 class, agreeing with the homology of the circle.

Rips filtration. This was originally implemented by Morozov in his Dionysus software [8], and several variants with theoretical guarantees are investigated in [11]. In this situation, $\mathbf{X}_0 \subseteq \mathbf{X}_1 \subseteq \dots$, and $r_0 \geq r_1 \geq \dots$. An example can be found in fig. 4.6. At the time of this writing, the computational bottleneck in this experiment is in a dense linear algebra implementation of the quiver algorithm, which grows cubically with the size of the point cloud. This is due to the large H_0 dimension at the end of the diagram, and we expect performance will improve significantly if replaced by a sparse computation.

4.5.2 Bivariate Nerve

Another application of zigzag homology is to investigate the relationship between algebraic features in nerves of two or more covers. This question arises naturally when generating covers algorithmically, where one might wonder how sensitive the Nerve is to choices that might be made or randomness in initialization. A related question is the stability of witness complexes to the choice of landmark set, and a *bivariate witness* complex was proposed in [3] and subsequently investigated by Tausz [12].

Definition 4.5.2.1. *Given two covers \mathcal{U}, \mathcal{V} of a space X , the bivariate cover is the fibered product $\mathcal{U} \times_X \mathcal{V} \subseteq \mathcal{U} \times \mathcal{V}$. Explicitly,*

$$\mathcal{U} \times_X \mathcal{V} = \{U \times V \mid U \in \mathcal{U}, V \in \mathcal{V}, U \cap V \neq \emptyset\}$$

and $\mathcal{U} \times \mathcal{V}$ can be identified as the intersection $U \cap V$ to form another cover of X . We will denote

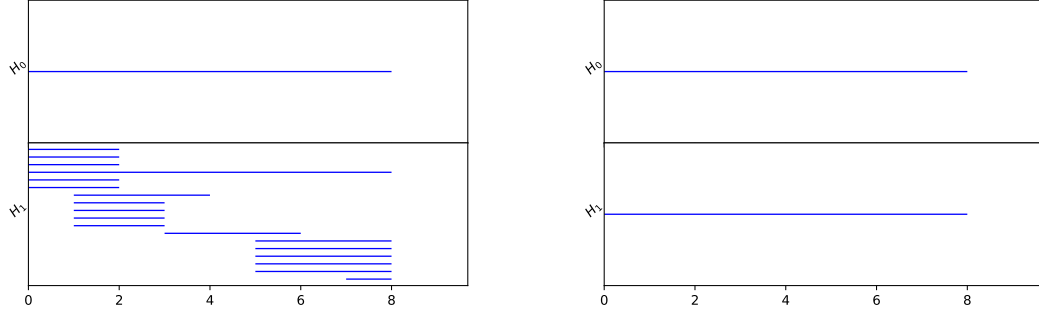


Figure 4.7: Zigzag barcodes of bivariate Nerve diagram on 5 covers of 500 points on the unit circle. Covers are computed by selecting 20 random landmarks. Left: each point is assigned to closest 2 landmarks. Right: each point is assigned to closest 3 landmarks. Note that both diagrams have single long bars in dimensions 0 and 1, agreeing with the homology of the circle.

the nerve of $\mathcal{U} \times_X \mathcal{V}$ as $\mathcal{N}(\mathcal{U}, \mathcal{V})$.

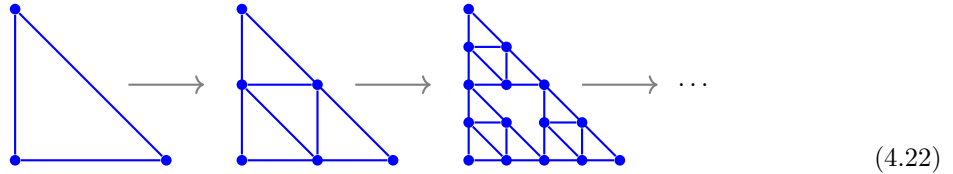
Due to the product structure on $\mathcal{U} \times \mathcal{V}$, there are projection maps $p_U : \mathcal{U} \times \mathcal{V} \mapsto \mathcal{U}$ and $p_V : \mathcal{U} \times \mathcal{V} \mapsto \mathcal{V}$. These maps extend to simplicial maps on the nerves:

$$\mathcal{N}(\mathcal{U}) \xleftarrow{p_U} \mathcal{N}(\mathcal{U}, \mathcal{V}) \xrightarrow{p_V} \mathcal{N}(\mathcal{V}) \quad (4.21)$$

We will consider covers based on landmark sets. Briefly, we choose a landmark set $\mathbf{L} \subset \mathbf{X}$, and assign each point in \mathbf{X} to the k -nearest landmarks. An example is performed in fig. 4.7, where covers of a noisy circle are created. We see that if each point belongs to only 2 sets that while there is a single long H_1 class, there are also many short-lived H_1 classes since there are no non-empty 3-way intersections to fill in small holes. However, if every point is assigned to 3 sets, then there is a single H_1 class that persists the length of the diagram.

4.5.3 Sierpinski Triangle

Our final application is an example of how persistent homology may be used with more general cell maps. We'll consider a sequence of spaces converging to a Sierpinski triangle:



Where each map above sends vertices to the vertex in the same location in the image to the

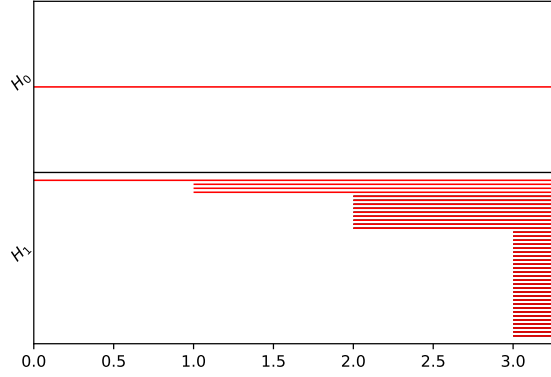


Figure 4.8: Persistence barcode showing 4 iterations of the sequence in Equation (4.22).

right. Then each edge is subdivided in the image, introducing a new node and two new edges, and additional edges are introduced to fill in new interior triangles.

The maps are not simplicial because each edge is subdivided at each iteration. One could also consider a filtration of simplicial complexes based on the finest Sierpinski mesh, but the advantage of the mapping construction is that it is easy to add another iteration of the subdivision to the end of the sequence without recomputing every space. We compute the persistence barcode in fig. 4.8. There is a single connected component, and the k th iteration (starting at $k = 0$) adds another 3^k H_1 classes.

Chapter 5

Applications of Zigzag Homology for Deep Learning

5.1 Neural Networks

Neural networks have emerged as a dominant model for machine learning in recent times. They have shown great success in prediction problems involving unstructured data such as images and natural language. Although the basic setup of the model and training is very straight forward to understand, the reason behind their success on these data sets is not very well understood. Thus it remains largely a form of technology that can be used to obtain higher performing models, but no clear theoretical road map on how to improve them or build upon them. This has resulted in the field largely composed of results obtained from trial and error exploration.

In this chapter we will provide a very quick introduction to deep learning and show how we can use persistence and zigzag homology to produce input features for our neural networks. This allows us to control what information is used by the neural networks to do the classification. This level of control helps us produce neural network models that generalize far better than just plain neural network models. We demonstrate this on the MNIST data set [7]. We also show that zigzag persistence gives us more expressive power than standard persistence homology. Thus illustrating that deliberately designing the architecture of the model so that it has certain properties can improve generalization performance.

5.1.1 Neural Network Architecture

The actual specification of the neural network model is usually fairly straight forward. The usual approach is to compose a set of functions chosen from a set of standard neural network components.

Definition 5.1.1.1. *A neural network is a function $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_h}$ specified by a sequence of layer functions $\{L_i\}$ for $i = 0, 1, \dots, h-1$, where each layer function $L_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_{i+1}}$ is a continuous non-linear function that is differentiable almost everywhere.*

$$y = L_{h-1}(L_{h-2}(\dots L_0(x)\dots))$$

There are many types of neural network layers, the most commonly used one is the fully connected layer. The fully connected layer consists of two operations, the linear part and the activation function.

$$a_{i+1} = F(W_i a_i + b_i)$$

Here a_i is considered to be the activations of the previous layer with a_0 being the input itself. The weight matrix W_i and bias vector b_i form the trainable parameters of the layer. The function F is called the activation function and is usually chosen to be a component wise non-linear function, so that stacking multiple fully connected layers will not result in a linear model. The most commonly used activation function is the rectified linear unit or ReLU activation.

$$F(x) = \max(x, 0)$$

This function is preferred because it has been shown to perform well in practice. It is considered to be a good activation function as it is mostly linear in its domain, apart from the non-linearity at 0.

When a neural network is used to tackle a classification problem, the final layer is usually chosen to be the softmax layer.

$$S(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

This ensures that the output behaves like a probability function, where all the values are between 0 and 1 and it sums to 1. Thus the entire model can be constructed by stacking fully connected layers of various sizes. For example, a 5 layer neural network can be constructed with the hidden layer sizes as follows : $a_1 = 100, a_2 = 200, a_3 = 200, a_4 = 200, a_5 = 10$. The number of hidden neurons usually increase up to a value and shrink to the output size, this is done to ensure that the neural network is expressive enough to handle the dataset.

5.1.2 Neural Network Training

In order to train a neural network on a dataset, we must first specify a loss function. A loss function is a continuous function that evaluates a particular set of parameters on a dataset. The lower the value of the loss function, the better the performance of the model on the dataset. We will only be looking at classification problems in this dissertation, so the outputs will be 0 or 1 depending on whether it is the correct class. The most commonly used loss function for classification is the cross

entropy loss.

$$J(y, \hat{y}) = \sum_i y_i \log(\hat{y}_i)$$

Here y is a vector encoding the true classification output using one-hot encoding, the correct output is 1 while the rest are 0. \hat{y} is the output of the model.

Once we have a loss function, training is usually done by minimizing the loss function on the dataset using one of the many available gradient descent algorithms. All such algorithms require the computation of the gradient. This can be achieved using automatic differentiation. All standard neural network libraries provide this capability. The neural network computation is structured as a computational graph, where each node performs an elementary operation. This allows us to apply the chain rule recursively on the graph, using the gradient for each component to go backwards in the graph. Thus the algorithm starts at the output node and recursively works its way backward to the input, the resultant algorithm is known as back-propagation.

Once the gradients are obtained, the weights are updated using one of the optimizer algorithms. The simplest optimizer is stochastic gradient descent. In this method, the dataset is sampled in batches, without replacement. The batch sizes can be varied to optimize performance, but is usually chosen to be a small number like 32. The gradient is computed on the batch and a single gradient step is taken to update the weights.

$$\hat{w}_i = w_i - \alpha g_i$$

Here w_i is one of the parameters, \hat{w}_i is the updated value, g_i is computed gradient and α is the learning rate. The learning rate is another hyper-parameter that can be optimized over to improve the training. Typical values for α range from 10^{-4} to 10^{-2} . Once we have gone through the dataset completely, 1 epoch of training is said to have occurred. Training is usually repeated for multiple epochs until the loss function is sufficiently low.

5.2 Application of Persistence and Zigzag Homology to MNIST

Our goal is to use the algorithms of persistence and zigzag homology to look at the digits of MNIST in a topological manner. So we take as input a particular MSNIST image which is 28 by 28 in size and produce as output a set of barcodes. The first step in this process is to convert the image into a simplicial complex. This is done in two stages, we first construct a graph from the image and then we take the flag complex produced by clique-closure. The graph is constructed by considering all the non-zero pixels as nodes. We then add an edge between the nodes corresponding to neighbouring pixels. We include diagonal pixels as neighbours, thus every pixel can have at most 8 neighbours. This graph now represents the topology of the digits, thus 6 and 9 would not be distinguishable in this form, thus to include such orientation information, we will add a filtration.

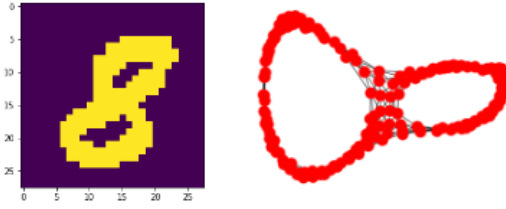


Figure 5.1: A MNIST digit and the graph implied by its pixels.

We can choose as the filtration value the pixel value in the image. This alone will not capture sufficient information to classify the digits, because orientation information is lost, leaving the digits 6 and 9 still indistinguishable. We need a procedure to add back the orientation information. This can be achieved by using various functions over the image as filtration values. First we define 8 directional functions

$$g_\theta(x, y) = \cos(\theta)x + \sin(\theta)y \quad \theta = 0, \pi/4, \pi/2, \dots, 7\pi/4$$

These functions are shifted and scaled so that in the domain of the image, they range from 0 to 1. If $I(x, y)$ is the input image, then the filtrations are given by

$$f_{\theta_i}(x, y) = I(x, y)g_{\theta_i}(x, y)$$

These functions simulate the effect of sweeping over the digits along the four cardinal directions. There are a total of 8 such sweeps, each filtration producing a different barcode.

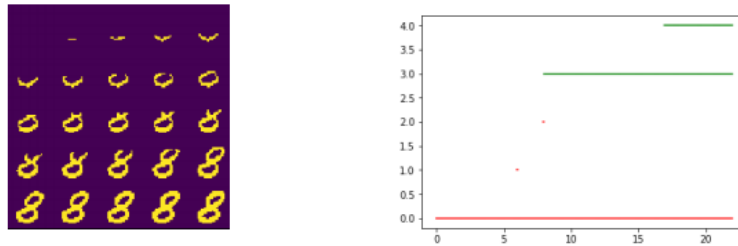


Figure 5.2: The sequence of images in the filtration and the corresponding barcode

In figure 5.2, we have an example of an image of the digit 8 being swept from the bottom to the top. Thus for every filtration we can produce a sequence of births and deaths for H_0 and H_1 , we can represent these pairs in a barcode plot. Here we draw a line in red for every pair in H_0 , the line goes from the birth to the death. Thus the X-axis represents the filtration value. The pairs from H_1 are represented in green.

We can now interpret this barcode plot for the digit 8 as follows, since there is overall one connected component, there is one long red bar. The digit 8 has two loops so we have two green bars. Since we are sweeping from the bottom, we close one loop first and close the second loop later, thus one of the green bars is born a little later than the other.

Thus in this way we can create persistence barcodes for all the eight different directions and this should capture the topological essence of the digit in the image. Similarly, the filtration can be used to create zigzag barcode as well. In the case of zigzag homology, we apply a windowed sweep. Instead of using level set filtration along the eight directions, we use an interval of fixed size. As this interval is swept along different directions, simplices are added and removed to create a zigzag diagram.

5.2.1 Featurization

Since we wish to train a neural network on these topological features, we need to convert them to a form that a neural network can process. The main problem here being that the number of pairs of births and death can vary depending on the input, but we need a finite representation to feed to a neural network. There are many featurization techniques in the literature such as persistence landscape [1], however we opted for the symmetric polynomial featurization.

$$F(m, n) = \sum_{i=1}^{N_f} (b_i + d_i)^m (d_i - b_i)^n$$

We basically construct a matrix of numbers of size N_f by N_f . The (m, n) element of this matrix is given by the formula above. It is essentially a polynomial function of the births and deaths summed over each of the features, thus giving a finite representation. We repeat this for H_0 and H_1 and also for each direction. We use $N_f = 5$, thus we have $25 \times 2 = 50$ features for each barcode plot and since we have 8 directions, we have a total of 400 features. Since the powers of the filtration values can be very large we apply the function $\log(1+x)$ on each value of the feature. We also subtract the mean and divide by the standard deviation computed over the entire training dataset.

5.2.2 Neural Network Model

Now that we have a finite size input which is 400 for each image, we build a simple 9-layer neural network with the following number of hidden units : [400,600,600,100,100,100,100,100,10]. We use cross-entropy loss with a softmax classifier.

Different architectures were tested but this was the one with the best performance. Training proved to be very hard, as most runs stopped improving after reaching 50% to 60% validation accuracy (training accuracy was similar). To improve the performance various techniques were employed which proved to be very effective in improving performance.

The first technique that helped in improving performance was triangular learning rates schedules. If we use only small learning rates, the loss always gets stuck in bad local minima, however large learning rates also performed poorly, what was found to perform the best was to use small learning rates initially then quickly change to large learning rate then wait until plateau and then reduce the learning rate again.

The second technique used to improve performance was to train on the failing cases only. Multiple initializations were analyzed and it was found that the examples they fail on were essentially the same set. Thus we first trained on the failing set only, then after getting reasonable performance we switched to the full dataset. At the end we again alternated between failing set and full set to squeeze out the last bit of performance.

Such ad-hoc training techniques are required, due to the highly non-linear nature of the objective function. Since we are using featurized barcodes as input data, some difficulty in training is expected. However once a system is established, successful training becomes repeatable.

5.3 Experiments and Results

Many different architectures and hyper parameters were tested. The performance was very poor when fewer than 9 layers were used. We initially used only 2 sweep directions, left to right and top to bottom, but the performance was not satisfactory. With persistence homology features, the techniques discussed in the previous section had to be employed to improve the performance. However the best performance was obtained when we used the zigzag barcodes. The table displays some of the representative results for the various different experiments performed.

Model	Training accuracy	Test accuracy
2-sweep-9layer	46%	45%
8-sweep-9layer	94%	82%
zigzag-9layer	99%	94%

Figure 5.3: Table summarizing the performance of models

Zigzag persistence homology features numbering 400 were created for 55 thousand training examples from MNIST. The test set had 5 thousand examples. We used a batch size of 1024. Simple stochastic gradient descent was used as ADAM did not perform very well. The learning rate was initially low at 0.001, increased to 0.01 for 2000 iterations and decreased back to 0.001. All initializations were not successful. Many of the initializations were stuck in local minima where two digits such as 6 and 8 or 0 and 6 were classified to the same class, and no amount of extra training would move it out of this local minima. Thus such runs were abandoned and restarted with a new initialization. Figure 5.4 shows the loss graph and confusion matrix for a successful run with the best performance.

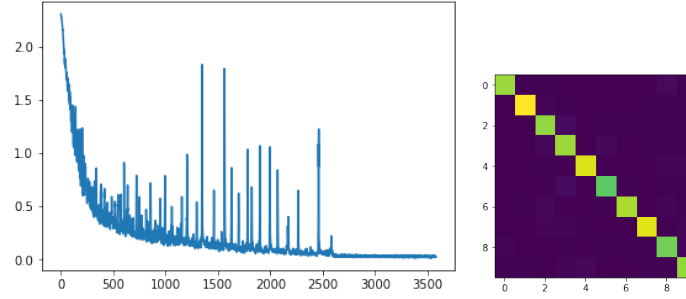


Figure 5.4: Loss graph and confusion matrix of the best model

In order to understand the benefit of using topological features in training the model, the model was tested with some hand created images of digits which a human would classify easily but a standard neural network might have trouble with. A small dataset of size around 50, consisting of very non-standard images of digits were created. The performance of the model was 70% on this dataset without any retraining. Figure 5.5 contains some of the examples that were classified correctly

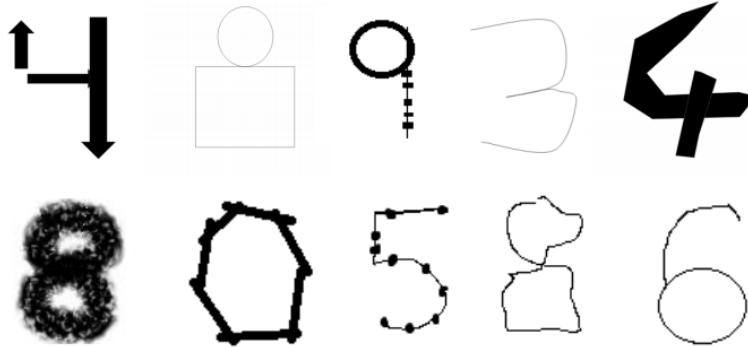


Figure 5.5: Qualitative results - Some examples that were correctly classified

As we can see, the model does not have trouble classifying correctly when the line width is varied and other distracting features are added. The model also works on arbitrary size images as the zigzag homology features can be computed in a scale invariant way. Although the model is not perfect in classifying such images, it is surprising to see the extent of generalization we can achieve without additional training.

5.4 Discussion

Thus we have shown how MNIST images can be featurised using techniques from TDA. Models trained on such features can generalize to images that standard neural network models fail to classify correctly. Switching from persistence homology to zigzag homology greatly improved performance, thus showing that the added flexibility of using zigzag can be used to enrich the features of the model. However even the best performing zigzag model only had 94% test accuracy. The main source of the error was found to be in the miss-classification of the digits 9 and 4. It turns out that the digit 9 and 4 in casual handwriting can be very similar, except for the difference in curvature in a part of the digit, with the digit 4 being sharper while the digit 9 is more rounder. Such distinctions in curvature seem to be hard to detect using topological features.

Bibliography

- [1] Peter Bubenik. The persistence landscape and some of its properties. *arXiv: Algebraic Topology*, 2018.
- [2] G. Carlsson, A. Dwaraknath, and B. J. Nelson. Persistent and Zigzag Homology: A Matrix Factorization Viewpoint. Preprint: <https://arxiv.org/abs/1911.10693>, 2019.
- [3] Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.
- [4] Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *SCG '09: Proceedings of the twenty-fifth annual symposium on Computational geometry*, page 247. ACM Press, 2009.
- [5] P. Gabriel. Unzerlegbare darstellungen I. *Manuscripta Mathematica*, 6:71–103, 1972.
- [6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 4th edition, 2013.
- [7] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- [8] Dmitiry Morozov. Dionysus2. Software available at <https://www.mrzv.org/software/dionysus2/>.
- [9] Bradley J. Nelson. *Parameterized Topological Data Analysis*. Ph.D. dissertation, Stanford University, 2020.
- [10] Steve Y. Oudot. *Persistence Theory: From Quiver Representations to Data Analysis*, volume 209 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2015.
- [11] Steve Y. Oudot and Donald R. Sheehy. Zigzag zoology: Rips zigzags for homology inference. *Foundations of Computational Mathematics*, 15(5):1151–1186, 2015.
- [12] Andrew Tausz. *Extensions and Applications of Persistence Based Algorithms in Computational Topology*. Ph.D. dissertation, Stanford University, 2012.