Data Engineering and Analytics
(International Master's Program)

Technische Universität München

Master's Thesis

Solving eigenproblems with Neural Networks

Nino Mumladze

Data Engineering and Analytics (International Master's Program)

Technische Universität München

Master's Thesis

Solving eigenproblems with Neural Networks

Author:	Nino Mumladze
Examiner:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Assistant advisor:	Dr. Felix Dietrich
Submission Date:	July 15th, 2021

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

July 15th, 2021 Nino Mumladze

Acknowledgments

I am very thankful for my supervisor, Felix Dietrix, for valuable input and insights shared throughout the whole process of working on the thesis.

Abstract

A lot of problems that arise in machine learning or pattern recognition boil down to solving eigenproblems $Ax = \lambda x$, with respect to x and λ. Tasks such as dimensionality reduction (PCA, Fisher's discriminant), Spectral clustering or data representation (Laplacian, Hessian eigenmaps or diffusion maps) are all based on calculating eigenvectors and eigenvalues of a matrix. There are various approaches for finding the spectral decomposition of a matrix. As finding the roots of a characteristic polynomial of a matrix becomes computationally infeasible in higher dimensions, there are only special cases where calculating eigenvalues exactly is possible in a finite number of steps. In general, algorithms for finding eigenvalues and eigenvectors are iterative, such as the power method, inverse method, Rayleigh quotient method, QR method and provide numerical approximations as opposed to exact solutions. As the sizes of matrices in the industry increases, it becomes important to solve eigenproblems as efficiently as possible, using methods that are fast, accurate and feasible even for large amounts of data. Recently, neural network based approaches have been proposed for this problem. The researches demonstrated that their approach could successfully solve linear algebraic systems with relatively short training time. In this thesis, we are going to tackle eigenproblems with Artificial Neural Networks (ANN) and compare the results with standard solvers in terms of accuracy, efficiency, etc. We demonstrate the accuracy of obtained eigenvectors by solving the heat equation.

Contents

1 Introduction

Applications of eigenproblems are everywhere. From computing higher powers of a matrix, to solving problems in physics or quantum chemistry. A lot of computer science tasks such as image restoration or information retrieval or clustering also boil down to finding eigenvectors and eigenvalues of a matrix. Developments of eigensolvers, therefore, benefit a lot of fields that rely on spectral properties derived from their data.

The earliest work that solves eigenproblems dates back to 19th Century, when Jacobi came up with a process of diagonalization, an iterative method for finding eigenvalues and eigenvectors of a symmetric matrix. However, it did not gain popularity until 1950's, when first modernized electronic computers were introduced. Since then, numerical algebra became a very actively researched topic and several theoretical developments have made it possible to grow computational techniques too. [13] states that around 40% of average 60 papers per year published in scientific SIAM Journal on Matrix Analysis and Applications (SIMAX), was about eigenvalue problem research.

To state the eigenvalue problem, it involves finding non-trivial solutions to the following equation for a matrix A:

$$Ax = \lambda x$$

This can be re-written as:

$$Ax - \lambda x = (A - \lambda I)x = 0$$

Eigenvectors and eigenvalues come in pairs. Applying a linear transformation A to its eigenvector w_i can at most change it by a scalar factor, so it only gets stretched out or squeezed, but does not change direction. The corresponding eigenvalue, often denoted by λ_i, represents the scaling factor of the vector in the new subspace. (λ_i, v_i) is then an eigenpair of matrix A.

Since we are searching for a non-zero vector x that, multiplied by a matrix, becomes zero, we know that the determinant of that matrix should be zero, so that the transformation $A - \lambda I$ squishes the space to a lower dimension.

$$det(A - \lambda I) = 0$$

This is called a characteristic equation. We can find eigenvalues of A by solving this equation. To do this algebraically, it is known that for polynomials of degree higher than 4, we do not have a closed form algebraic equation to find its roots, and that's why other iterative numerical methods are employed. However, except for some special cases, explicitly determining and solving this equation is often numerically highly unstable, since large perturbations of the roots could be caused by even very small perturbations of the

polynomial coefficients.

This obstacle lead to development of other techniques for solving eigenproblems. One very important iterative method for this task is power method, and it became a basis method for many modern eigensolvers. It is the simplest iterative method that converges to the biggest eigenvalue. The procedure involves repeatedly multiplying matrix A by some vector v. With enough iterations, the component of vector v corresponding to the direction of the dominant eigenvector becomes significantly higher in magnitude than all other components. This method, however, is no longer competitive with modern eigensolvers and not used in standard solvers, but is a basis for more powerful QR method, and methods developed of Arnoldi and Lanczos.

There are mainly two types of eigensolvers: direct (Gaussian elimination) and iterative (conjugate gradients [19]). Direct solvers differ from iterative solvers in that they produce the solution in fixed number of steps, while iterative solvers produce increasingly accurate approximations to the solution. Direct solvers operate on the transformed matrix to get the eigensystem that is equivalent but easier to solve, usually transforming the matrix to its tridiagonal form, while purely iterative methods try to find the eigenvalues and eigenvectors of a lower-dimentional subspace of the matrix that represents it well. The latter algorithm is highly favorable due to its property of producing accurate and useful solution estimate when stopped early, but also requires finite number of iterations to terminate. When matrices are sparse and large, and we are interested in only a small fraction of the eigenspectrum, iterative solvers, such as Lanczos method, Jacobi-Davidson methods or the Jacobi method are widely used. However, if we want to find the whole spectrum of the matrix, then direct solvers are a more feasible choice.

The cg-method was proposed initially as an alternative to solve well-defined problems, but in the early 70's the researchers started utilizing the method for other problems as well and now it is heavily used for unconstrained optimization problems or singular matrices. It is a good tool approximate solutions to linearized Partial Differential Equations (PDEs), or to solve large and sparse systems of linear equations, where matrix modification methods, which required a lot of fast computational capabilities on matrices, become unpractical and have a higher time($\mathcal{O}(n^3)$) and memory complexity.

In order to evaluate and compare the novel approach of solving eigenproblems with Neural networks, first it is important to overview proven methods that are currently industry standard for finding eigenvalues and eigenvectors.

Linear equation solvers that are widely used today are implemented in libraries such as PetSC [1], LAPACK (using BlAS subroutines), are typically operating on full matrices instead of batches. Even though they have highly reliable and accurate, fast results and are widely used today, novel approaches have still been developed to employ non-linear neural networks to solve the same problems, since they can operate on small batches of data and also give us a function output in the end, which can be applied to any previously

unknown input.

Solving eigenproblems with complex non-linear functions seems like over-complication of a problem. In reality, eigenproblems sometimes get too big to solve with direct methods and that's where non-linear solvers offer their benefits. Non-linear methods could come up with solutions that are more compact, and also are suitable for training on batches of the data, and therefore have less rigorous memory requirements.

In [34] time recurrent neural networks were proposed to obtain the smallest and largest eigenvalues of a symmetric matrix and recently, more and more machine learning techniques have been brought up to solve certain eigenvalue problems. [35] demonstrated possibility of using reinforcement learning to find eigenvalues and eigenvectors of a k-sparse eigenvalue problem. [11] recently showcased employing game theory for PCA, which is a type of eigenvalue problem. It reformulates PCA as an Eigengame, a competetive multi agent game. Each player controls an eigenvector, and they are punished for getting too similar to other players, while the better they explain variance in the data, the higher the score of their reward function. The nash equilibrium of this game gives the solution of PCA and it can be found by using gradient descent for each independent player to optimize for their own reward function. This approach made it possible to perform spectral clustering or graphs or find principal components of massive datasets [12].

The goal of this research is to investigate non-linear, neural network based solver to solve eigenproblems and apply computed eigenvalues and eigenvectors to solve a partial differential equation for heat diffusion and see if it gives a viable solution. Solutions to partial differential equations can sometimes be computed analytically, but usually numerical methods are employed to solve them. Physics informed Neural network based approaches have also been brought forward to solve PDEs [14], that use supervised learning and therefore rely on good quality training data, but they are not useful for solving eigenproblems.

Since neural networks are stochastically trained, it is unlikely that they will provide competitive performance to standard solvers, however, using neural networks could provide us with better scallability and other benefits, such as a possibility to use it as one of the steps in future eigensolvers.

Other approaches that solve eigenproblems do not provide a continuous function that can be used to map points to their spectral embeddings, instead they approximate eigenfunctions by discretizing the manifold and then computing the eigenvectors as approximations to eigenfunctions. Manifold learning, for example, needs to be defined firstly in terms of a method for local parametrization of the data and secondly a way of combining the local parametrization to get an effective global parametrization of the whole manifold. In theory, neural networks, having universal approximation capabilities for any arbitrary function, could solve this problem more naturally and directly, without having to discretize the problem, but instead using a training procedure to adjust the shape of the function to resemble the one that describes the true function the best. The complexity however, is that training such a neural network could pose challenges in itself. For example, the eigenfunctions need to be orthogonal to each other, and ensuring orthogonality when

training a neural network with mini-batch procedure, is difficult.

The network of choice is SpectralNet [27], which takes a deep learning approach towards finding eigenvectors and creates spectral embeddings of input data by embedding data points into eigenspace of the Laplacian associated with the graph constructed from the data.

2 State Of the Art

In this section we introduce existing algorithms and specific implementations related to finding spectral properties of matrices. Section 2.1 introduces spectral clustering, a well-known technique for dimensionality reduction that utilizes spectrum of a the similarity matrix to find lower level representation of the data prior to clustering, while preserving information about underlying density and geometry of the data. Section 2.2 introduces diffusion maps, another approach for dimensionality reduction, targeted specifically to the Laplacian. Subsection 2.3 introduces Datafold, a software written in python that interprets time-series data as dynamical systems and point clouds as geometrical structures with operator-theoretic models. It, among many other machine learning methods, implements above-mentioned diffusion map algorithm using standard eigensolvers. 2.4 describes existing neural network architecture designed for spectral clustering, which conversely to standard eigensolvers, does stochastic gradient optimization steps towards learning eigenfunctions of a matrix.

2.1 Spectral Clustering

Spectral Clustering [17] is an unsupervised learning technique that has its roots in graph theory. It is a very powerful clustering tool compared to k-means or Gaussian mixture models. On one hand, the latter techniques are very efficient, but can only be successfully be applied to limited cases where data points are grouped in convex shapes. On the other hand, spectral clustering can identify non-convex clusters, and it does so by analyzing spectrum of special matrices constructed from the data set. Even though it is better applicable to clustering problems with more complex point cloud shapes, it is less computationally efficient and hard to scale up to large datasets. Similarity matrix, which is necessary for performing this type of clustering, is quadratic to the data size and often becomes challenging to analyze without employing a supercomputer with huge memory capabilities. The space complexity of the algorithm is $O(N^2)$, and the time complexity $O(N^3)$, and there has been a lot of effort put into reducing this complexity.

Adjacency Matrix is a matrix, constructed from the data, important for this method. First, every type of data is translated into a graph, this can be done by seeing points in our datasets as collection of nodes and edges that connect them. Adjacency matrix is a matrix representation of the graph, and it is constructed by setting A_ij element to one, if

ith and jth nodes are connected to each other by an edge, and 0 otherwise. Affinity matrix is another important matrix, which is similar to adjacency matrix, however the distinction between them is that the edges between the nodes are weighted according to some weight function, instead of having value 1 for all edges.

In order to determine which nodes have edges between them and which ones do not, there is a need for a notion of closeness. Typically, either matrices are constructed by considering number of nearest neighbours, that is chosen arbitrarily and edges are assumed between node and it's nearest n neighbours. Alternatively, euclidean distance between the nodes is measured and a cut off parameter ϵ is chosen, which determines how close the two nodes should be in order to have an edge between them, so if $||x_i - x_j|| < \epsilon$, there is an edge between ith and jth nodes, and if the distance is larger, the two nodes are disconnected. ϵ-neighourhood graphs are geometrically motivated, however, it is difficult to choose the a good ϵ parameter and it often leads to several connected components in the graph. n-nearest neighbourhood graphs are less geometrically intuitive but it is easier to choose the number of nearest neighbours for the given dataset. Since we would be computing eigen-decomposition of those matrices, it is important to note that both ϵ-neighbourhood and n-nearest neighbourhood adjacency matrix is sparse and symmetric.

Then, to get affinity matrix from adjacency matrix, the edge are weighted depending on the euclidean distance between the nodes, often as a weight function a gaussian (heat) kernel is used:

$$W = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right), \tag{2.1}$$

where σ is a standard deviation parameter that controls bandwidth of the gaussian filter. It represents typical distance between the points in the dataset. The bandwidth is heuristically selectde, typically by considering mean or median value of distances between the data points.

Affinity and Adjancency matrices have zeroes on the diagonal, since nodes are not connected to themselves. In case of an undirected graph, Adjacency and affinity matrices are symmetric, as the edges have two directions, and $A_ij = A_ji$. Affinity matrix can also be viewed as a diffusion operator of a graph.

Degree Matrix D represents the number of outgoing/incoming edges for each node. If the graph is undirected, incoming and outgoing edges are the same, since the edge goes both ways. The Degree matrix is a diagonal matrix, and the degree of ith node is in D_ii cell. Every other entry is zero. We can compute the degree matrix from adjacency matrix, by summing up every row and making the end result a diagonal matrix.

Unnormalized Graph laplacian is another matrix describing graph data. It is a discrete analogue of the laplacian operator and measures how a function differs at a point compared to its nearby points. It is a symmetric and positive semi-definite matrix that encodes information about the connectivity of the graph, as described in detail in [7]. It's

eigenvalues are real and non-negative. The number of 0 eigenvalues of the graph laplacian indicate number of connected components in the graph. The importance of second eigenvalue of graph laplacian, Fiedler's value is also known since 1973 when Fiedler discovered it encodes information about connectivity of the graph. 2.1 visualizes the graph laplacian.

It is calculated by:

$$L = A - D$$

The positive-semidefiniteness of the matrix tells us that it's eigenvalues are all non-negative and real valued. The smalles eigenvalue is 0 corresponding to the constant eigenvector of 1's.

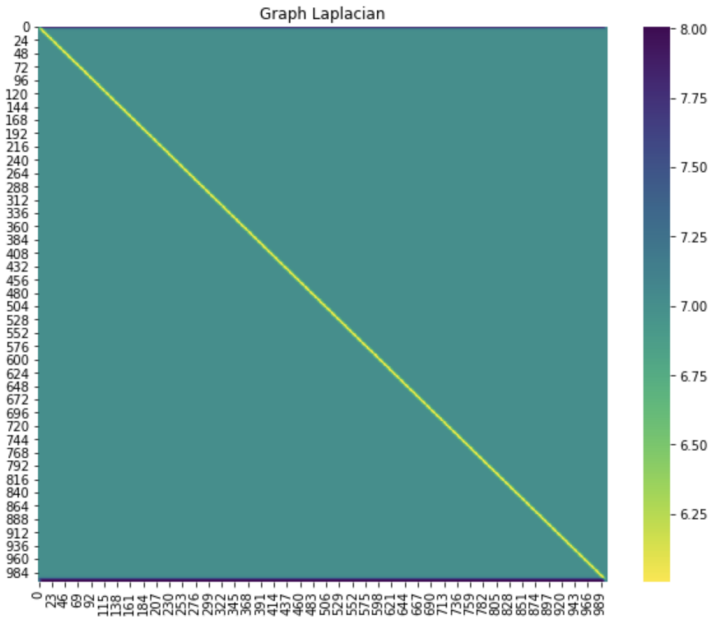


Figure 2.1: Graph laplacian for 1000 points on a circle

Normalized Graph laplacian is often used instead of graph laplacian for spectral clustering and spectral partitioning. It is important to note that normalized and unnormalized laplacians share the same eigenvalues. However, using normalized graph laplacian for finding eigenvectors provides better consistency guarantees when number of data points increase. In spectral graph theory, there is a notion of cut, which represents number of edges between clusters. If the cut is small, the formed clusters have low intra cluster connectivity. The main difference in using normalized versus unnormalized laplacian is that while unnormalized graph laplacian optimizes optimal ratio cut in a graph, the normalized graph laplacian optimizes for normalized cut.

$$Ncut(A,B) = \frac{cut(A,B)(Vol(A)+Vol(B)}{Vol(A)Vol(B)}$$

,

where $Vol(X)$ is sum of the weights of edges in the set X

$$Vol(X) = \sum_{i \in X} \sum_{j} W_{ij}$$

,

and cut(A, B) is the sum of weights of edges connecting A and B partitions of the graph $cut(A,B) = \sum_{i \in A, j \in B} W_i j$, where W is a weighted adjacency matrix.

Then, to perform spectral clustering, the spectrum of the chosen graph laplacian (normalized or unnormalized) is found. The second smallest eigenvector is used, either by considering the sign of the vector components or performing K-means on the components. 1 shows the pseudo-code for performing spectral clustering.

[30] analyzed consistency of spectral clustering and showed superiority of normalized method from statistical point of view. The normalized graph laplacian is defined by:

$$\mathcal{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

For optimal cluster recognition, the Laplacian should approximately resemble a diagonal block matrix, where each block is represents a cluster and each sub block gives us information about shape of the cluster. Other useful properties derived from this matrix are used for spectral clustering, such as spectral gap, difference between first largest eigenvalues, that conveys information about density of the node distribution or number of eigenvalues before the first big gap between eigenvalues, which usually represents a number of clusters in the graph, and fiedler value and fiedler vector which can be used to partition a graph. In general, spectral properties of this graph are often analyzed and studied, and as the matrices are sparse and symmetric, eigensolvers targeted towards large and sparse matrices are of great importance for spectral clustering applications.

Algorithm 1: Spectral Clustering

Result: Cluster assignments

$A = SquaredDistances(X)$;

$W = GaussianKernel(A, scale)$;

$L = D - W$;

$L = D^{\frac{-1}{2}} L D^{\frac{-1}{2}}$;

$eigenvalues, eigenvectors = eigsh(L)$;

$U = eigenvectors[:: k]$;

$clusters = KMeans(k).predict(U)$;

In [3] Spectral clustering showed significant improvement over other clustering techniques such as standard K means, due to its ability to find non-convex cluster shapes. The

main trick that causes this improvement is the change of the representation of the abstract data points to a representation that, due to the properties of the graph laplacian, represents the cluster shapes in the data in an improved way, and therefore is simpler to detect for the K-means algorithm. There have been studies [25] analyzing stability or consistency of spectral clustering, as well as it's usage towards problems in many specific fields. [29] describes various fields spectral clustering is used for, such as image segmentation, entity resolution, educational data mining and also presents recent improvements in time complexity or parallel computing to better equip the method to be highly scalable to large datasets.

2.2 Diffusion Maps

Exploratory Data analysis often requires transforming points lying in a higher dimensional space into lower dimensional representation that still offers meaningful insight about the data. When analyzing large scale simulations of dynamical systems that change over time, it is important to identify slow variables that capture evolution of the system in a longer timeframe. This is a dimensionality reduction problem that can be solved in many ways. One of the more robust and versatile methods of dimensionality reduction is manifold learning. Manifold is a lower dimensional set of points embedded in a high dimensional space. For example, a subspace could be non-linearly embedded in a space, but have linear instristic geometry. 2.2 illustrates that.

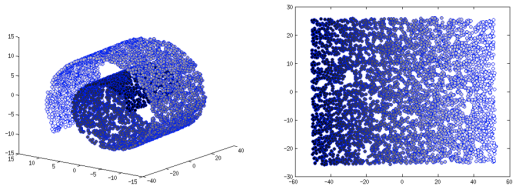


Figure 2.2: Non linearly embedded intristicly linear manifold

By choosing the right, either linear or non linear transformation for the points, the intristic complexity of the manifold could be preserved in just a few dimensions. Manifold learning, similarly to spectral clustering, is a non-linear dimensionality reduction method, therefore it can be more expressive than famous PCA, due to it's ability to preserve underlying structure of not only linear manifolds, but also non-linear ones. One of the algorithms for manifold learning, basis of which is that data is only artificially high-dimensional, is diffusion maps. It works by computing the distance between points not in euclidean space, which is only locally relevant, but over the low-dimensional manifold that the data is assumed to reside in.

Diffusion maps [6] are a non-linear unsupervised learning method, that learn meaning-

ful geometric representations of the data that is useful for dimensionality reduction. This framework is computationally inexpensive and unifies many ideas from machine learning and is connected to eigenmap methods and spectral graph theory [5]. It results in a representation of X that is constructed from integrating on local geometry of the graph data. The method approximates diffusion distances which then can be used for analyzing connectivity within the graph, clustering or ranking. It is a robust method that is insensitive to noisy perturbations.

Diffusion maps construct random walk on the data, illustrated in 2.3, where each transition step has it's associated probability. This random walk, gives us a way to translate the geometrical structure of a manifold into a matrix. Given a dataset X and a kernel k that represents prior definition of local geometry of X, diffusion maps algorithm construct a reversible markov chain on X, known as normalized graph laplacian matrix. The kernel is called a diffusion kernel and its shows connectivity between two nodes, and a gaussian kernel (2.1) here is a common choice too. The kernel defines local neighbourhood where euclidean distance is still relevant, and outside of this distance, the values quickly go to 0. The kernel has two properties:

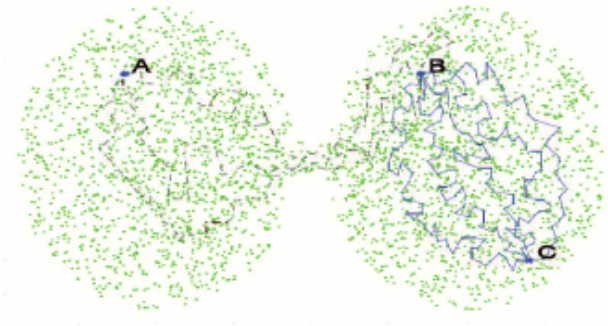


Figure 2.3: Random walk on the manifold

- It is positivity preserving $k(x, y) >= 0$.
- It is symmetric, so k(y, x) = $k(x, y)$.

The idea then, is that running the chain forward in time, or taking higher order powers of this diffusion matrix, can be used to perform multiscale geometric analysis of X. Since powers of the operator are of interest, spectral theory is employed to use eigenvalues and eigenvectors and efficiently compute the powers of the kernel. Kernel eigenmap methods have proposed the idea that eigenvectors of the kernel matrix can be viewed as coordinates in the data set. Given those, we can compute diffusion distances between points, as well as create a mapping to embed points in a euclidean space, where diffusion distance of those points is equal to euclidean distance.

Difussion maps are connected to kernel PCA methods, which is a general approach for all kernel based manifold learning methods [16]. These methods have offer major improvements over classical dimensionality reduction approaches: They preserve local structures and are non linear. Non linearity of these methods are essential, since typically points rarely lie on linear manifolds. Preserving locality is another very important benefit. As we know, in higher dimensions, long distances become meaningless, since due to the curse of dimensionality, similarity measures break down. Preserving locality then works against skewing the results towards those high distances.

Algorithm 1 Basic Diffusion Mapping Algorithm

INPUT: High dimensional data set X_i, $i = 0 \dots N-1$.

1. Define a kernel, $k(x, y)$ and create a kernel matrix, K, such that $K_{i,j} = k(X_i, X_j)$.
2. Create the diffusion matrix by normalising the rows of the kernel matrix.
3. Calculate the eigenvectors of the diffusion matrix.
4. Map to the d-dimensional diffusion space at time t, using the d dominant eigenvectors and -values as shown in (9).

OUTPUT: Lower dimensional data set Y_i, $i = 0..N-1$.

Figure 2.4: Basic Diffusion Map algorithm from [26]

Thus, diffusion maps employ eigenvectors corresponding to the first few eigenvalues of markov matrices to embed data in lower dimensions. Diffusion maps represent the data as edges and vertices, where edges are weighted by the heat kernel with parameter ϵ. One of the downsides of the diffusion maps algorithm is that this parameter is data dependent and therefore a new scaling parameter has to be chosen for each new dataset.

In [2] mathematical justification was provided for using the first few eigenvectors as approximations of eigenfunctions of the Laplace-Beltrami operator, when data is uniformly sampled from a low dimensional manifold. Laplace-beltrami operator is a generalization of the Laplacian to Riemannian manifolds. Riemannian manifold is a collection of points that locally resembles euclidean space but globally is different and has defined Riemannian metric, which measures the length of the curves as well as the length of the tangent vectors.

However, it is important to note that different scales yield different embeddings. The mapping to eigenspace therefore is not unique, rather, there is a family of diffusion maps, which also results from different normalization of the markov chain. A parameter α can

specify the amount of influence of the density in the infinitesimal transitions of the diffusion. The influence of density is maximized if we set αa to be zero, which is equivalent to graph laplacian normalization with gaussian weights and is best used for spectral clustering. In case of $\alpha = \frac{1}{2}$, markov chain normalization then leads to approximation of the Fokker–Planck operator, which is especially useful when analyzing limiting behaviour of high dimensional stochastic systems. Lastly, another important operator can be approximated if we set the α parameter to one, Laplace-Beltrami (heat) operator, which, regardless of the density of the data, can be used to analyze geometry of the point cloud.

[26] showed connection between the heat equation and random walks on graphs and discussed computing eigenvectors and eigenvalues of the heat diffusion equation directly, which is what diffusion maps algorithm does. Especially when the domain is high-dimensional and complex, solving the heat equation becomes very hard, but this way it can still be approximated.

2.3 Datafold

Datafold [21] is an open-source software package that enables inferring underlying geometrical structures from large-scale datasets. In comparison to equation driven approach, datafold provides data-driven models for analyzing point clouds and time series on manifolds, which givers users more flexibility to analyze wider range of systems, even the ones where the system of equation is either intractable or even unknown.

Datafold's API is templated from popylar machine learning library scikit-learn. It is highly customizable and adjustable to many machine learning problems and abstracts away the complexity of defining algorithms that solve the problem of representing those complex or simple data structures. It's infrastructure is based on using data format structures widely used in scientific context, namely Numpy and Scipy arrays. It works on sparse matrices, which provides more robust and scalable solutions.

Datafold views data points as point clouds embedded over a manifold, and for global parametrization of those point clouds, it provides an efficient implementation of diffusion maps. This particular implementation of the algorithm stands out with it's ability to handle sparse kernel matrix. Additionally, it can allows choosing an arbitrary kernel. It can be utilized to infer geometrically meaningful structures from data, such as approximate eigenfunctions of operators: Laplace-Beltrami, Fokker-Plank, Graph Laplacian.

Instead of approximating Laplace-Beltrami operator, the choice of parameter $alpha = 0$ gives us an estimation of eigenvectors of the graph laplacian of the gaussian kernel with the specified bandwidth (sigma parameter of the gaussian) that can be optimized for each manifold. For stability of eigenvector computations, the gaussian kernel is normalized to obtain a conjugate transformation, a symmetric conjugate kernel with the same spectral properties, and then eigenvalue decomposition happens with scipys methods that operate on sparse matrices to offer faster solutions with less memory requirements.

Datafold uses ARPACK library, which is an industry standard, robust and scalable method

for solving eigenproblems. Datafold implements a wrapper of eigensolver calls and chooses optimal subroutine depending on the matrix and it's density. Hermitian matrix is matrix that is equal to its conjugate transpose and sparsity of a matrix is determined by having mostly non-zero entries. The final solver is chosen between 4 subroutines displayed in table 1.

	Hermitian	Non-hermitian
Sparse	**scipy.sparse.linalg.eigh**	**scipy.sparse.linalg.eig**
Dense	**scipy.linalg.eigh**	**scipy.linalg.eig**

Table 1: Scipy solvers for different types of matrices

Under the hood, datafold uses scipy's eigensolver, which, in turn, is a wrapper for ARPACK [22] software subroutines that use Implicitly Restarted Arnoldi Method to find the eigenvalues and eigenvectors of sparse matrices. For symmetric matrices, the this process becomes another Krylov subspace method, called the Implicitly Restarted Lanczos Method (IRLM). For dense matrices, scipy calls LAPACK, which uses a variation of QR algorithm. Scipy solver is selected based on number of eigenpairs to be computed, and often sparse solver is favoured over dense solvers, because often dense solvers find all eigenpairs which is a computationally costly operation when only solving for k eigenpairs.

The software provides a way to transform out of sample points and embed them in their respective spectral image by implementing Nyström extension, allowing the users to extend the diffusion map algorithm to unseen data.

Datafold also provides a way of choosing parameters for gaussian affinity, namely cut off and ϵ parameters in it's estimators class. Cut off represents a value of distance above which distance is treated as zero when computing the kernel matrix, it is estimated by choosing maximum distance between points and it's nearest k neighbours. ϵ represents gaussian kernel scale, which, for given tolerance parameter, is estimated this way:

$$\epsilon = \frac{\text{cut-off}^2}{-log(\text{tolerance})} \tag{2.2}$$

where by default, tolerance$= 1e^{-08}, k = 25$ and cut-off is estimated to be maximum distance from each node to it's 25th neighbour.

The parameters estimated from datafold are then used for computing the kernel matrix for SpectalNet too, which is going to be discussed next.

2.4 SpectralNet

Spectralnet [27] solves a problem of spectral clustering, which is unsupervised learning method that is widely used when working with graph data. It's primary goal is to find good clustering assignments to points in the data, where similar data points are grouped

together, while dissimilar points are in separate groups. So maximising intra cluster connectivity score while minimising inter cluster connectivity. If we consider random walk on a graph, Spectral Clustering can also be interpreted as searching for a partitioning of the graph, for which the probability of jumping between clusters is low and the random walk stays within clusters for the longest time. In essence, Spectral clustering performs K mean clustering on embedded data in eigenspace of normalized laplacian matrix, derived from pairwise similarities between data points. However, even though embeddings only require eigen-decomposition of a matrix, for large datasets direct computation of the eigenvalues is infeasible.

SpectralNet, is a neural network that uses deep learning for performing spectral embedding on the data. It addresses the shortcomings of traditional spectral clustering approaches: poor scalability and out-of-sample-extension. It's learning process is based on constrained stochastic optimisation, which enables scallability to larger datasets. The constraint, which is there to ensure orthogonality of the outputs, is implemented by a special purpose output layer, designed to make the output of the network orthogonal.

Out of sample extension problem is non-trivial problem, as showcased in [4], where 5 algorithms with no straightforward way to generalize over unseen data points, were extended to be able to embed new data points without recalculating the eigenvectors, due to the convergence of eigenvectors to eigenfunctions. Conversely, Spectralnet uses constrained stochastic optimization to learn a mapping function from input to output space, that can be generalized to map any point to it's difussion map, solving the OOSE problem. The function is represented as a feed forward deep neural network, and the added constraint is needed to ensure orthogonality of the eigenvectors and is achieved a linear layer, whose weights are set by the QR decomposition of it's inputs.

SpectralNet stochastic optimization process ensures scalability to large datasets. [18] demonstrated that spectral clustering could be reformulated as a constrained optimization problem and approximated in mini-batch training, relying on stochastic gradients to compute unbiased estimations of exact gradients with $\mathcal{O}n$ cost instead of $\mathcal{O}n^2$. At each iteration, this method uses only a fixed subset of columns of the laplacian to compute the gradients. Due to the stochastic characteristic of the training process, the orthogonality constraint is not strictly satisfied for the full dataset, however, shuffling the data and choosing different mini-batches gives a good approximation to orthogonal vectors.

The most interesting part in SpectralNet, is that, once trained, SpectralNet computes a map $F_\theta : \mathbb{R}^d \to \mathbb{R}^k$. It maps each input point x to an output $y = F_\theta(x)$. The spectral map F_θ is implemented using a neural network, and the parameter vector θ denotes the network weights.

The training of SpectralNet consists of three components:

- Unsupervised training of Siamese Network to learn an affinity given the input distance measure.

- Unsupervised learning of the map F_θ by optimizing a spectral clustering objective while enforcing orthogonality.
- Using K-means on the spectral embeddings to learning the cluster assignments.

The first and the last components are less important for this thesis. However, the second component, learning the function via the neural network is an important step that can be utilized to enhance or, in the future, substitute other eigensolvers.

End-to-end training approach for learning F_θ is summarized below in 2.

Algorithm 2: SpectralNet Training

Input: Data Matrix $X \subseteq R^d$, batch size m ;
Output: embeddings $y_1, ..., y_n, y_i \in R^k$;
while $L_{SpectralNet}(\theta)$ *not converged* **do**
  Orthogonalization step:
  Sample a random minibatch X of size m;
  Forward propagate X and compute inputs to orthogonalization layer $\tilde{Y}$;
  Compute the Cholesky factorization $LL^T = \tilde{Y}^T\tilde{Y}$;
  Set the weights of the orthogonalization layer to be $\sqrt{m}(L^{-1})^T$;
  Gradient step:
  Sample a random minibatch X of size m;
  Compute the m × m gaussian affinity matrix W;
  Forward propagate $x_1, ..., x_m$ to get $y_1, ..., y_m$;
  Compute the loss ;
  Use the gradient of $L_{SpectralNet}(\theta)$ to tune all F_θ weights, except those of the output layer;
end
Forward propagate $x_1, ..., x_n$ and obtain F_θ outputs $y_1, ..., y_n$;

SpectralNet training involves two steps, and happens in coordinate descent fashion, alternating between orthogonalization and gradient steps. For each step, different random mini-batch of size m is selected. Orthogonalization step adjusts weights of the last linear layer of the network, which is responsible for orthogonalizing its inputs, while the gradient steps uses backpropagation to update every other layer's weights except for the last one.

The loss for SpectralNet if we use unnormalized laplacian is:

$$L_{SpectralNet}(\theta) = \frac{1}{m^2} = \sum_{i,j=1}^{m} W_{i,j}||y_i - y_j||^2$$

It can be rewritten as:

$$L_{SpectralNet}(\theta) = \frac{2}{m^2} = trace(Y^T(D - W)Y)$$

Considering the orthogonality of the outputs, the minimum of the loss function is reached when columns of Y^T span the subspace of the first k eigenvalues of $D - W$.. It is defined as a sum over pairs of points from the dataset, which is unlike other loss functions, that typically sum over individual samples. Each summand is representative of relationship between the two chosen nodes.

SpectralNet represents neural network based approach to finding diffusion maps. It approximates diffusion map with major computational speedup, which is linear in dataset and model size. [23] used this diffusion map approximation as a prior for defining the manifold to overcome prior mismatch problem when training VAEs or GANs. SpectralNet does offer an advantage of not having a bias towards forming only convex shaped clusters, unlike other methods that use Gaussian priors, and then perform variational inference to find lower dimensional latent space, such as one described in [20].

SpectralNet approximates eigenvectors of the laplacian well enough to allow the network to cluster non-convex shaped point clouds, which still poses a challenge for other deep learning approaches trying to solve the same problem. It achieved state-of-the-art performance on Reuters dataset, which is a large corpus for text categorization, for which traditional spectral clustering is impractical. SpectralNet can be used to optimize for eigenfunctions of normalized or unnormalized laplacian, as well as another neural network, called "Siamese Net" can be employed to learn different affinity function between the data points to use instead of the standard gaussian affinity.

Other approaches have also been tried to tackle the problem of scalability of clustering methods. [28] successfully suggested reducing size of similarity matrix for spectral clustering to improve clustering result of k-means, which was achieved even for the datasets where Mcut [8] algorithm could not be performed. [9] used Nystrom extension, a technique to approximate eigenfunctions and solve OOSE, to first perform image segmentation for small subset of pixels and then extrapolate the results to full image.

It is important to note that other deep learning approaches for performing clustering directly optimize Kullback–Leibler (KL) divergence, and serve as an autoencoder that tries to encode target distribution into centroid-based prior probability distribution. One such example was described in [33], where spectral convolution and spectral transformer networks were built and used to perform 3D shape augmentation. With SpectralNet, we first get an intermediate result of data points transformed to their eigenspace and then perform clustering. This enables us to utilize eigenvectors of the laplacian for other tasks as well. Additionaly, since the gaussian kernel matrix is computed, we can compute eigenvalues corresponding to each eigenvectors, which would have been impossible with most other deep learning based spectral clustering approaches, since they do not compute the kernel explicitly.

3 Solving Eigenproblems With Artificial Neural Networks

In this chapter, first we are going to introduce the architecture of neural network used for solving the eigenproblem, with it's activation functions and choice of layers, define it's loss function and describe how the network tries to satisfy the orthogonality constrained. The next section, 3.2 introduces evaluation metrics used for evaluating the quality/accuracy of resulting eigenvectors, since we have an unsupervised learning problem and the loss or validation loss does not convey how well our network approximates eigenfunctions. Consequently, in 3.3 we introduce set up for our models and experiments. Firstly, SpectralNet was primarily used for clustering, and then to examine how well a neural network can estimate eigenfunctions and eigenvalues of the Laplacian. Moreover, the results of choosing hyperparameters for our training procedure are shown. 3.4 describes the process of actually computing the eigenvectors and eigenvalues with the neural network. 3.4 presents comparison of evaluations of two sets of eigenvectors, obtained from SpectralNet and Datafold. In 3.5, the results are applied to solve one of the test examples, eigenfunctions and eigenvalues are used to obtain the solution to the heat equation on the circle. The results are then compared in terms of how close the numerical approximation with the eigenvectors comes to the true exact solution computed analytically.

3.1 The Architecture of Neural Network of SpectralNet

Neural networks are a complex computing system that is used for various problems. Some problems are hard for humans, but incedibly easy for computers, typically the ones that involve a lot of calculations. However, pattern recognition is a task that conversely, is difficult for a machine, and typically neural networks are often employed for solving such problems. In the paper named "A logical calculus of the ideas imminent in nervous activity", published 1943, a neuro scientists described the first artificial neural network, inspired by human brain cells. Even though the building blocks of the network, neurons are incredibly simple and are only responsible for reading an input, processing it and returning an output, collective power of the neurons today, can solve some tasks with even better performance than a human. Neural networks are universal approximators and can theoretically be constructed and trained to approximate any arbitrary function, they are adaptive and can change their internal structure according to the data given to it. A typical neural network consists of layers of neurons, which represent the smallest computing units in the network. Neural network algorithm is defined by it's architecture (the way neurons are arranged), the loss function and the optimisation strategy. The loss function is a function for which neural network tries to achieve an optimum, using the optimisation strategy that we define.

Different neural network architectures have been developed for different kind of problems. The simples neural network, perceptron, could only solve linearly separable tasks. Later on, Convolutional networks were developed for image recognition, Recurrent neural networks were created for sequential data and are mostly used in Natural language pro-

cessing, and as the field of deep learning advanced, neural networks not only classified data, but became generators too. Moreover, different architectures have been brought up that serve the purpose of Spectral Analysis. SA-NET, [31] for example, is one such other neural network architecture, that uses spectral analysis layer to employ deep learning and analyze images for clustering. Following Han and Filippone ([18]), demonstrating that the spectrum of the laplacian can be computer with mini-batch training, another algorithm, called SpectralNet, was developed to approximate the eigenfunctions of the laplacian with a neural network.

For SpectralNet, the architecture is customizable. We can choose number of layers and number of neurons in the layers. 3.5 shows a typical fully connected neural network architecture with 3 hidden layers. For our problem, we also started with using a Neural Network with three hidden layers. With 128, 64, and 32 neurons, respectively. The output dimension is determined by the number of eigenvectors we want our network to compute, k. Since our output needs to be orthonormal, additional linear layer is added at the end, which performs orthogonalization of the output by computing QR decomposition via cholesky decomposition and transofrming input to be othogonal. After that, in order to use the eigenvectors, we have to normalize them to have norm 1 and range between -1 and 1.

The last linear layer, therefore, for input matrix X, where X^TX is full rank, performs cholesky decomposition:
$X^TX = LL^T$
where L is a lower triangular matrix with positive diagonal entries. Then the orthogonalization of the input can be obtained with:

$$Q = X(L^{-1})^T$$

So at each orthogonalization step, which is a step that happens during training process of the network, the weights of the last linear layer are set to be $L^(-1)^T$.

Each neuron in the network has it's own set of weights and biases, as well as an activation function, which indicates whether the neuron is active or not (propagates information forward or not). There are many activation functions available, and their choice typically depends on the task. For this problem, in our network architecture Elu activation function was used. SpectralNet's implementation did not use this activation function and keras layer interface was used to add Elu activation function to the network architecture. Elu is identical to famous RELU for positive inputs, but is smoother towards 0. This smoothness makes it better suited for optimization, since it is differentiable everywhere and therefore performs well during backpropagation. However, it is important to note that Elu tends to cause saturation problems when it's inputs are large negative values.

$$R(z) = \begin{cases} z & z < 0 \\ \alpha(e^z – 1) & z \geq 0 \end{cases}$$

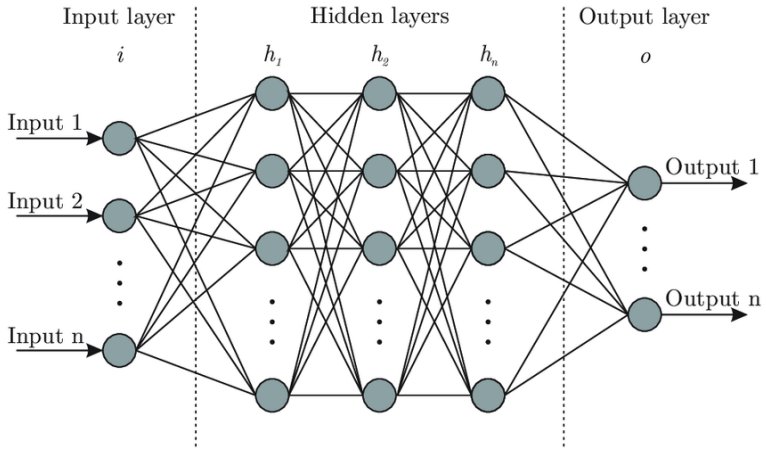


Figure 3.5: Architecture of Fully connected artificial neural network with 3 hidden layers

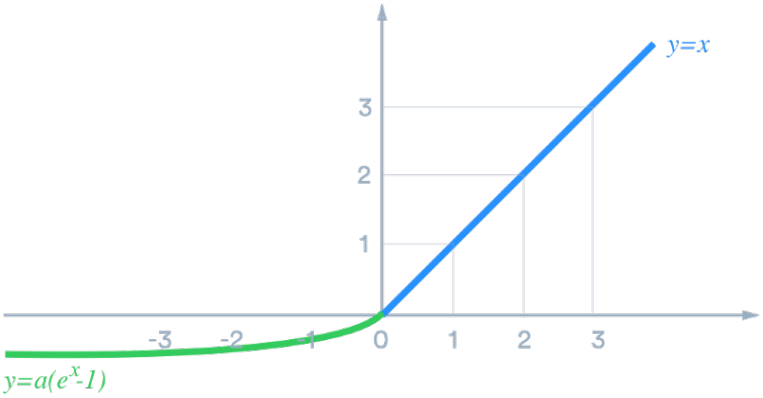


Figure 3.6: Elu activation function

The loss function of the network is set up to find eigenvectors of the graph Laplacian of gaussian kernel on the data. The optimization algorithm for SpectralNet is RMSPropOptimizer(Root Mean Square Propagation), which is a gradient-based optimization algorithm with adaptive learning rate, where each weight has it's own learning rate that gets adapted depending on the magnitude of the gradient. Gradients of functions such as neural networks often either explode (get too large) or vanish (become nearly zero) as the data propagates through the deep layers of the network, and RMSProp was designed to avoid this, by decreasing the step size for large gradients and increasing it for small gradients.

$$L_{SpectralNet}(\theta) = \frac{1}{m^2} \sum_{i,j=1}^{m} W_i j \left\| \frac{y_i}{d_i} - \frac{y_j}{d_j} \right\|^2$$

where $d_i = D_{i,i} = \sum_{j=1}^{m} W_{i,j}$, and this loss ensures training the network to optimize for

eigenfunctions of normalized graph laplacian $I - D^{\frac{-1}{2}} W D^{\frac{-1}{2}}$.

3.2 Evaluation Metrics

As we have already seen, we are performing unsupervised learning with our artificial neural network. When choosing optimal model and hyperparameters for the models for the supervised case, we rely on our test and validation data, for which we have labels for, and evaluate how well our network generalizes. However, theoretically, we don't have the eigenvectors of the matrices we are looking for, and thus we train our network in an unsupervised way. However, choosing the right model becomes trickier, and so we need other methods to essentially distinguish between poor and good performance of our model.

In order to evaluate quality of eigenvectors computed from the methods, it is useful to take a look into couple of metrics that will gives us capability not only to measure the change in loss of the neural network function, but also make sure that the network is indeed approximating the eigenvectors.

Grassmann distance is a measure that was used in SpectralNet in order to compute the quality of eigenvector approximation. Grassmann distance is calculated between subspaces of true eigenvectors and spectralnet output. It represents square root of sum of squared distances of sine angles between two subspaces.

$$d(A, B) = (\sum_{n=1}^{k} \theta_i^2)^{1/2}$$

, where θ_i is the ith principal angle between A and B.

$$\theta_i = cos^{-1}(\sigma_i(A^T B)$$

and σ_i is the ith singular value.

This measure, therefore, estimates similarity of the two sets of eigenvectors given the relative position of two subspaces spanned by them. Figure 3.7 from [32] visualizes the intuition behind grassmann distance.

Squared L2 distance was a measure that was used in SpectralNet in order to compute the quality of eigenvector by comparing norm of difference vector between datafold and spectralnet outputs. Where V_s and V_d are vectors computed from spectralnet and datafold respectively.

$$L_2(V_s, V_d)^2 = \sum_{i,j} (V_s ij - V_d ij)^2$$

For the estimation of quality of eigenvectors computed by neural networks in [24] the following measures were used also for our purpose.

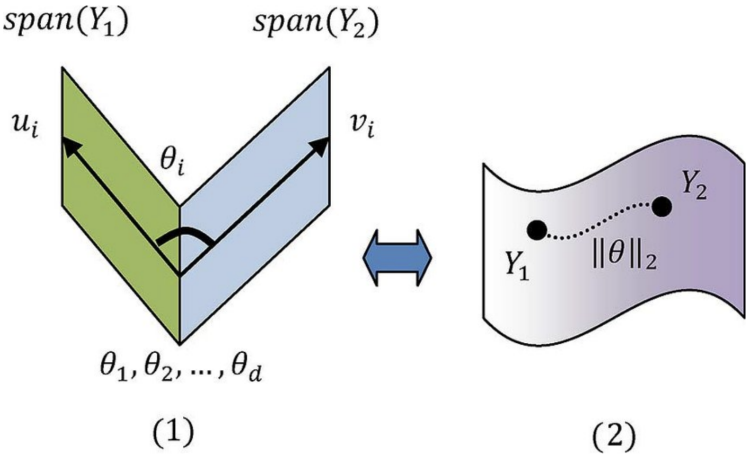


Figure 3.7: $span(Y_i)'s$ are a d-dimensional subspace in R. The distance between them can be measured by principal angles, $\theta_i's$., In the Grassmann manifold point of view, these subspaces are points on the manifold G(d, D). Then, the distance between these points is $||\theta||_2$.
1) Principal angle in R^D. 2) Grassmann distance in G(d, D).

Projection/Reconstruction error which measures how close the subspaces spanned by w_i's are to the one spanned by q_i's, where $i = 1..K$, the smaller the value of E, the closer the two sub-spaces are to each other.

$$E = \sum_{n=1}^{K} \frac{\boldsymbol{w_i}}{||\boldsymbol{w_i}||} - \sum_{i=1}^{K} \boldsymbol{q_i}\boldsymbol{q_i}^T \frac{\boldsymbol{w_i}}{||\boldsymbol{w_i}||}$$

The metrics above capture how close to sets of vectors are close to each other, however, extent of orthogonality is not captured in them. So additionally, orthogonality measure which is essentialy capturing the same thing as $Orth_max$ from [24] is calculated:

Orthogonality Measure to ensure that orthogonality constraint is met.

$$Orth = ||I - VV^T||$$

where V is matrix with computer eigenvectors. The lower the measure, the less the norm of the error between identity matrix and VV^T, since we know ideally they should be equal.

3.3 Models and Experiments

Experimenting with Spectral Clustering With SpectralNet

The model of neural network that was primarily used is SpectralNet, which provides a new way to perform spectral clustering via learning the spectral embedding of the graph

laplacian.

Prior to using SpectralNet to solve the heat equation numerically, the network was used to perform spectral clustering on one of sklearn's dataset, concentric circles with non convex decision boundary. The dataset can be generated specifying number of points, the noise parameter that is added to the data, and the factor parameter which controls the scale between inner and outer circle.

It is important to mention that the training was happening without any labeled data, as opposed to other examples listed in SpectralNet paper, where training always was either fully, or at least semi-supervised and the network had at least some labeled data. For our clustering experiments, the data provided to the network was only the points, without the cluster assignments.

It performed consistently well for recognizing the clusters. 1000 points from the dataset were sampled with a noise parameter 0.1 and a factor of 0.2. Without having to extensively search through for hyperparameters, it outperformed standard clustering with K-means, which, due to non-deterministicity of the algorithm, yielded different clustering assignments and did not recognize cluster shapes due to it's inability to recognize non-linear boundaries. However, standard spectral clustering and spectralnet both had resulted with correct cluster assignments over multiple runs. 3.8 Illustrates the circles where the coloring indicates each cluster an algorithm assigned a particular point to.

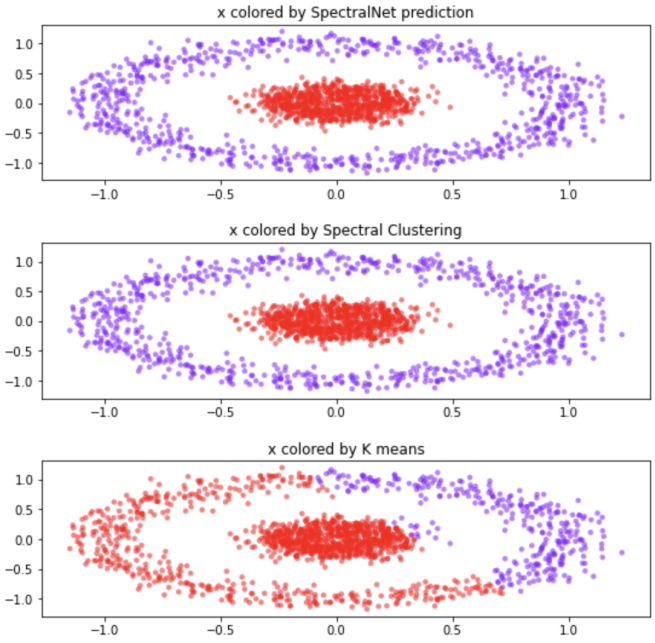


Figure 3.8: Results of SpectralNet, Standard Spectral Clustering and K-means clustering. Sampling 1500 points, with noise set to 0.1 and factor 0.2

However, changing the dataset a bit to have a different shape but still have similar boundary, changed the results of the network. The Figure 3.9 shows the cluster assignments generated from again 1500 samples, noise=0.05, but this time with a factor of 0.5. Here, SpectralNet is unable to correctly classify points into clusters without supervised learning.

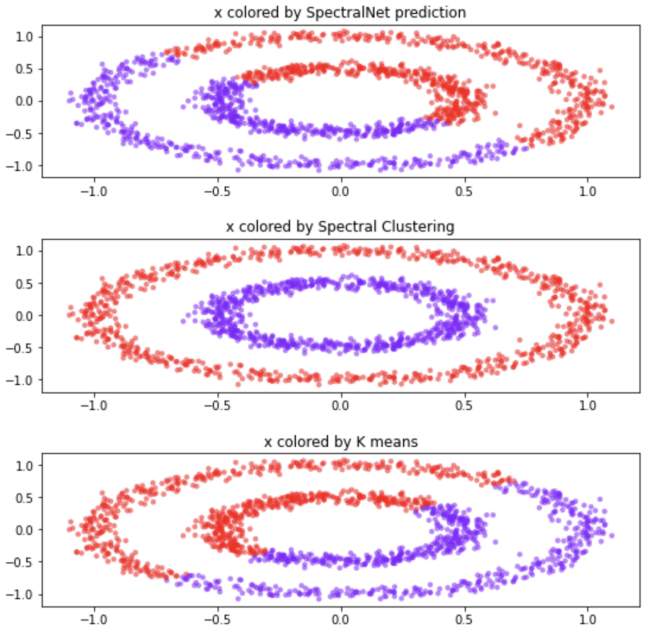


Figure 3.9: Results of SpectralNet, Standard Spectral Clustering and K-means clustering. Sampling 1500 points, with noise set to 0.05 and factor 0.5

Using SpectralNet to estimate eigenfunctions and eigenvalues of The Laplacian

As SpectralNet's loss attains optimum when Y is subspace of eigenfunctions of the Laplacian, we try to measure how good those estimations are, by comparing it with eigenfunctions computed by another standard method, that comes from the software package Datafold. Datafold, as previously discussed, performs manifold learning, which refers to learning underlying structure of a point cloud. This is a process of finding lower dimensional representation, and can be achieved with methods such as kernel PCA or Spectral Clustering. Datafold provides efficient implementation of "Diffusion Maps" model, which involves finding eigenfunctions of markov matrices defining a random walk on the data to obtain a new representation of the dataset, which is equivalently, optimizing for the same function as SpectralNet's loss.

The hyperparameters for datafold is only a scale parameter for the gaussian kernel, which, is estimated with the help of the same package. However, SpectralNet, apart from the scale parameter, requires a lot of additional settings.

As with any kind of neural network training, hyperparameter searching is done before determining the best hyperparameters for the given problem. In this case, grid search was performed to choose optimal learning rate, patience, dropout and number of epochs. In 3.10 the part of grid search results is presented. As optimal parameters, the 19th row was chosen, with learning rate $1e-4$, patience 30, number of epochs 70, and dropout to be 0.2 since it had lower accuracy and grassmann measure.

	lr	dropout	pt	epochs	norm	grassman	loss	p_error	val_loss
0	0.000050	0.100000	10	50	2.965858	0.003744	0.000642	0.152513	0.000012
1	0.000050	0.100000	10	70	2.872364	0.004460	0.000643	0.164316	0.000012
2	0.000050	0.100000	30	50	2.841706	0.002991	0.000640	0.156320	0.000012
3	0.000050	0.100000	30	70	3.854709	0.002911	0.000641	0.131622	0.000012
4	0.000050	0.200000	10	50	1.213847	0.010887	0.000639	0.133445	0.000012
5	0.000050	0.200000	10	70	2.840428	0.003788	0.000640	0.142116	0.000012
6	0.000050	0.200000	30	50	3.220545	0.004437	0.000637	0.158539	0.000012
7	0.000050	0.200000	30	70	2.865897	0.002808	0.000641	0.150705	0.000012
8	0.000050	0.300000	10	50	3.378475	0.004979	0.000642	0.106256	0.000012
9	0.000050	0.300000	10	70	2.857502	0.006773	0.000641	0.177283	0.000012
10	0.000050	0.300000	30	50	3.471703	0.004387	0.000637	0.201407	0.000012
11	0.000050	0.300000	30	70	3.398394	0.003894	0.000637	0.165689	0.000012
12	0.000100	0.100000	10	50	2.837335	0.002676	0.000641	0.095646	0.000012
13	0.000100	0.100000	10	70	2.897338	0.001869	0.000638	0.093450	0.000012
14	0.000100	0.100000	30	50	2.917193	0.003197	0.000637	0.140679	0.000012
15	0.000100	0.100000	30	70	3.345472	0.004521	0.000641	0.155239	0.000012
16	0.000100	0.200000	10	50	3.460719	0.002348	0.000639	0.101426	0.000012
17	0.000100	0.200000	10	70	3.231347	0.001810	0.000637	0.129633	0.000012
18	0.000100	0.200000	30	50	2.984432	0.001534	0.000638	0.135119	0.000012
19	0.000100	0.200000	30	70	3.444175	0.001128	0.000636	0.146699	0.000012
20	0.000100	0.300000	10	50	3.636574	0.002814	0.000639	0.112152	0.000012
21	0.000100	0.300000	10	70	2.971574	0.003580	0.000642	0.115539	0.000012

Figure 3.10: Hyperparameter Searching, grid search results

Apart from normal hyperparameters of the neural network such as learning rate, patience, dropout and number of epochs, our problem also needs to use a scale parameter for the gaussian kernel construction. choice of scale parameter controls bandwidth of the gaussian kernel. Both SpectralNet and Datafold provides a way to estimate the scale to be used.

SpectralNet takes median distance of each node to it's 25th neighbour as a heuristic for the scale parameter, while datafold's heuristic for the scale is desribed in equation 2.2.

Their optimal scale varies, but in order to make sensible comparisons of two sets of eigenvectors, our experiments used datafold's scale parameter for constructing the kernel matrix for SpectralNet. Additionaly, the scale for spectralnet was standard deviation,

while for datafold it was used as a variance. So our parameter for the network was the following:

Since datafold uses it's ϵ parameter to compute the kernel matrix the following way:

$$W = exp(\frac{-D^2}{2 * \epsilon})$$

where D is a matrix of pairwise euclidean distances.

And SpectralNet exponentiates by squared of it's input scale as seen in equation 2.1, therefore:

$$\sigma_{SpectralNet} = \epsilon^{1/2}$$

The data for our example comes from a simple circle. The dataset is created by sampling θ, which represents an angle and is between 0 and 2π N=10 000 times, then multiplying radius by cosine and sine of θ and getting the x and y coordinates on the circle, respectively. Figure 3.11 shows the plotted generated circle. Then two approaches are used to find k eigenvectors corresponding to the graph normalized Laplacian for this data.

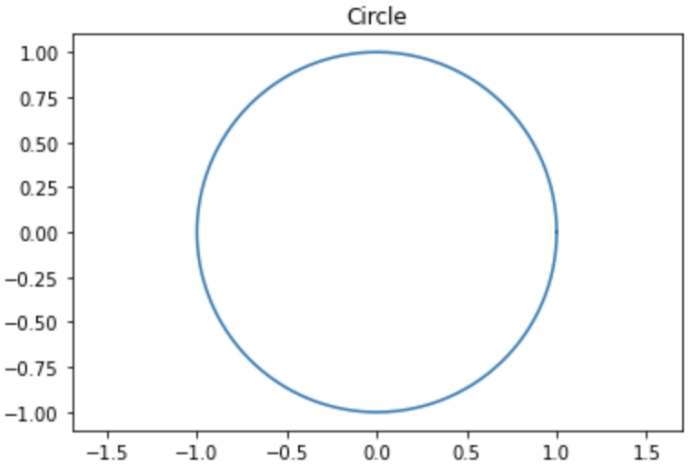


Figure 3.11: Unit circle with 10000 points

Two models differ from each other in terms of time needed to compute the eigenvectors. Datafold uses direct solver and therefore is much faster compared to spectral net. Figure 3.12 demonstrates this advantage, where the setup and compute times are compared for two methods.

Choosing the mini-batch size

The optimization happens on mini batches. Mini batch size is another hyperparameter that can influence stability of the training process and speed of the learning, as well as

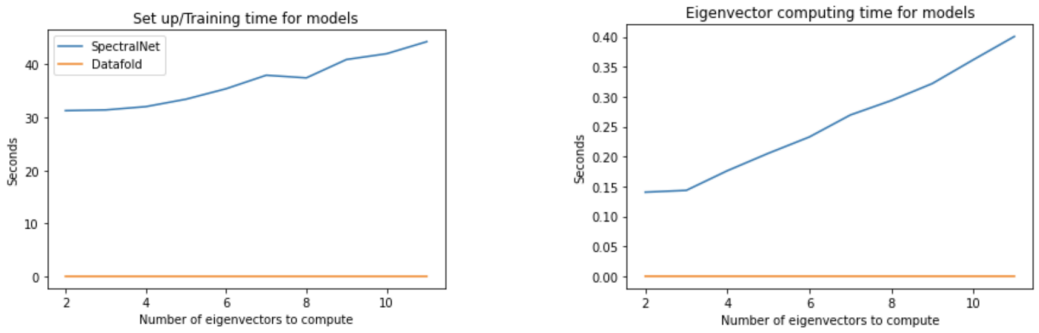


Figure 3.12: Change in Training and Compute time with increasing K

Parameter	Value
Optimizer	RMSprop
Initial Learning Rate	$1e^{-4}$
Batch Size	256
Batch Size Orthonorm	512
Weight Initializer	Glorot uniform
Number of Epochs	70

Table 2: Chosen Hyperparameters for network training.

generalization ability of the network. Neural network training process involves updating the network learnable weights according to the gradient of error estimate, which is calculated on a given batch. The larger the batch, the better it represents the overall data and the more accurate the gradient updates would be. During SpectralNet training process, the learning iteration chooses two different mini-batches. One for updating the weights of the last layer, and one for the rest of the network. The batch size, therefore greatly influences how orthogonal are the resulting eigenvectors to each other. It should be large enough to still be able to produce orthogonal outputs, without diminishing benefits of mini-batch training. Figure illustrates that the larger the mini-batch, the more orthogonal the output of the network become. For our testing, as a batch size for orthonormal layer, we used batch size of 512, because even though increasing batch size reduces the orthonogonality error and grassmann distance, using large batch-sizes is impractical (since generally smaller batch sizes are favoured, for faster iterations) and the network performed better when orthonorm batch size was closer to the batch size for the rest of the network.

Finally, all chosen hyperparameters for network training is summarized in 2.

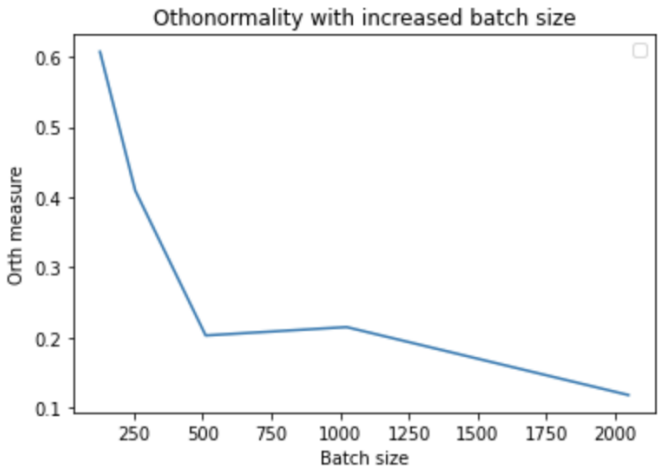


Figure 3.13: Change of Orthogonality with Batch Size

Adding more complexity to the network

When searching for neural network architecture, it is a common practice to start with small amount of layers and neurons and add more learnable parameters later if necessary. Since 3 layered neural network has less learnable parameters than a deeper and wider neural network, it has less complexity and expressability. By making the network wider or deeper, we estimated whether this change improved performance of the network. Unfortunately, increasing either the width or the depth of the network did not result in better quality eigenvectors.

3.4 Computing Eigenvectors and Eigenvalues

In order to compute eigenvectors, we first train SpectralNet on our data. Plotting the loss after training gives us intuition about whether our network converged to something or diverged and did not find good enough approximation of the function we are optimizing for. In our case, it can be seen on figure 3.14 that the loss converges and goes down from initial starting point, so we know the network is wired up correctly.

The loss we are optimizing for, however, does not capture quality of our eigenvectors. For that, we also consider grassmann distance and how it changes over time, depicted on 3.15.

After training, predict "spectral" embeddings on the data, which gives us approximations of eigenfunctions of the laplacian. Firstly, the output is normalized to make the eigenvectors orthonormal. Then, in order to sort the eigenvalues in terms of importance, eigenvalues need to be calculated. It is known that the quality of eigenvalue estimation depends on how separated each eigenvalue is to its closest neighbouring eigenvalues, and similarly, eigenvectors are also sensitive to the direction of other eigenvectors correspond-

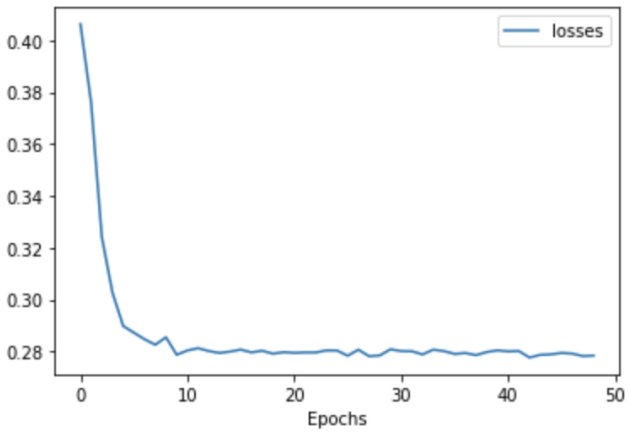


Figure 3.14: Change of loss over epoches

ing to neighbouring eigenvalues. The more separated the values are, the less sensitive the eigenvalue estimation is to numerical perturbations.

In order to calculate the eigenvalues, the kernel matrix itself is needed. It is computed by fixing the indexes of the shuffled dataset, running the Tensorflow graph on the whole dataset after training, to evaluate the kernel, and then predicting the eigenvectors for the same exact order of the data. The output of the network is then multiplied by the row-normalized kernel and divided by the output again, in order to get N approximations of the k eigenvalues.

After plotting the distribution of eigenvalues, the median of the distribution is considered as an approximation to the eigenvalue itself in order to compare with values computed by datafold. Another heuristic, the mean of the distribution was compared to the median and the norm of the error between datafold's output and the median of the approximations was the minimum of the two. 3.16 vizualizes the reasoning behind the chosen heuristic. This procedure gave us a very comparable approximation for eigenvalues of the laplacian matrix, difference between the norms of eigenvalues came close to 0.

Estimating Eigenvalues with spectral net gave pretty accurate results. Figure 3.17 plots the 5 smallest eigenvalues of the laplacian, which can be computed by reversing the gaussian filter

$$\lambda = \frac{2 * log(value)}{\epsilon}$$

where epsilon is the used scale parameter and $value$ is the eigenvalue computed with the gaussian kernel.

It is important to note that when we increase the size of our last layer (want to compute more eigenvectors), to compute more than 11 eigenvectors, the input to cholesky decomposition becomes invalid due to numerical errors, and matrix $X^T X$, where X is the input

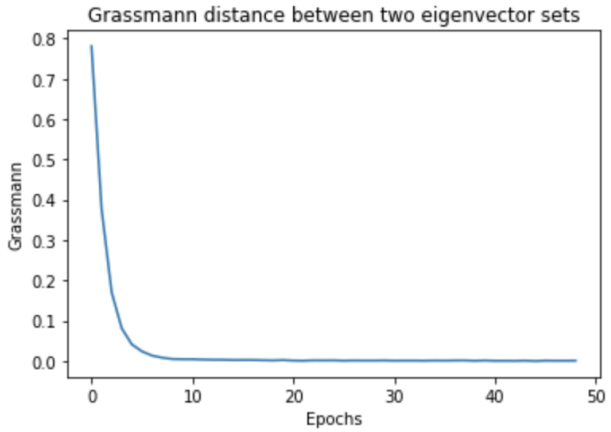


Figure 3.15: Change of Grassmann distance between subspaces spanned by Datafold's and SpectralNet's approximated eigenvectors during training.

to the orthogonalization layer at the end of the network, becomes singular.

Additionaly, it is important to note that the more the number of eigenvectors we want to compute, the less exact they are. Figure 3.18 shows the rise in norm of difference of eigenvalues computes from two methods, although it remains close to zero even when estimating 10 eigenvalues. This is not suprising since we are using the whole dataset to get estimations for eigenvalues, whereas the eigenfunctions are estimated on smaller batches.

The results from datafold package are plotted below, they represent sine and cosine functions, as expected. We plot each eigenvector against our sampled variable θ.

Contrastly, the eigenvectors from spectralnet do not look as accurate. They still look like periodic functions and their frequency gets higher and higher we compute more eigenvectors, but are arguably less resembling the cosine and sine functions than datafold output. Also, spectralnet only provides eigenvectors and eigenvalues are not computed, therefore the order of the eigenvectors is unknown and it is not guaranteed that they are sorted. Although the spectralnet loss function optimizes for a subspace in which the constant vector is contained, but spectralnet fails to estimate the constant vector exactly, and the difference in eigenvalues is also often more prominent for the smallest eigenvalue (zero).

Moreover, the neural network output is not deterministic, therefore, each time we train a network, even with the choice of same parameters, we get a different result. For this particular neural network, the results are not stable, and this can be visualized in the 3.19, where a network was trained with same parameters and norm of error between the eigenvalues (after scaling them back) and the heat equation solution was measured, and it has large deviations.

One of the reasons of this instability could be network's sensitivity to it's initial weights. First off, replacing the standard choice of Glorot uniform (Xavier) initialization with another weight initializer, which initialized weights to be orthogonal to each other was tested,

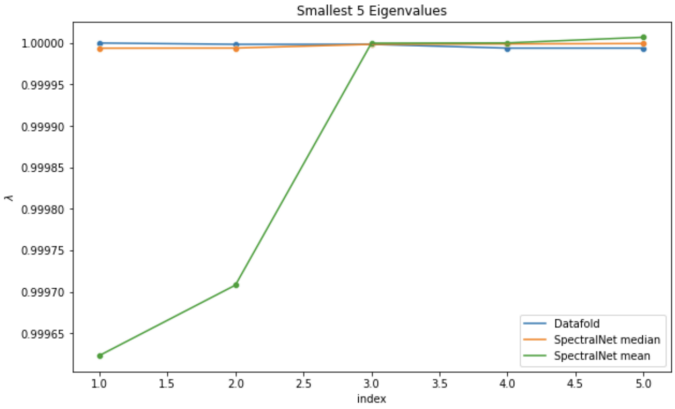


Figure 3.16: 5 Smallest eigenvalues of the laplacian, estimated by datafold, and from spectralnet with mean and median heuristics

it's performance was even less predictable, however, in some cases solution to the heat equation was approximated much better. To test whether the output was inconsistent due to the sensitivity towards choice of weights, a seed parameter was passed to a weight initializer, in which case the network training process became much more reliable. To combat this sensitivity towards initialization, batch normalization layers were added in between the layers to stabilize the training process. In the end, Xavier initialization without batch-normalisation layers performed in the most stable way shown in figure 3.20.

3.5 Test example: The heat equation on a circle

Heat equation Background

The Laplacian is a differential operator given by the divergence of the gradient of a function on Euclidean space. It often appears in differential equations that describe physical phenomena. One such example is the heat equation, which describes diffusion of the heat in a region. The solution to the heat equation gives the distribution of temperature in a region as a function of space and time for specified temperature at the boundaries, the initial distribution of temperature, and the physical properties of the medium.

Partial differential equations are useful when a process can be easier described through the process of change. The heat equation is an partial differential equation that describes how heat diffuses over time, mainly, the rate of heat transfer on a given points depends on the temperature of the point itself.

Let $g(\theta)$ be the initial temperature on a thin unit circular rod. Let $u(t, \theta)$ be the tempera-

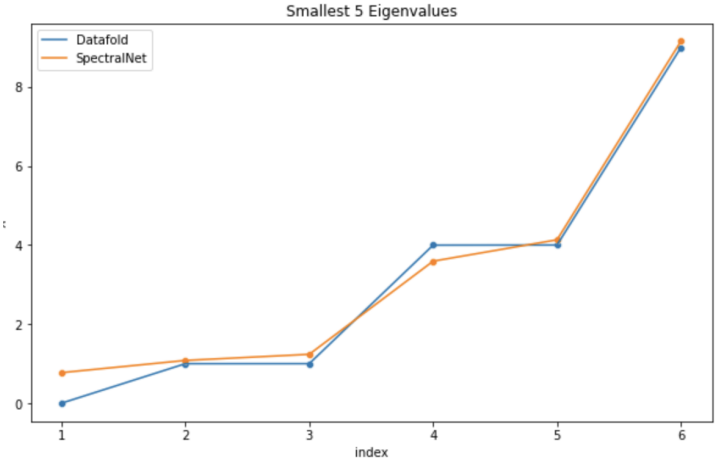


Figure 3.17: 5 Smallest eigenvalues of the laplacian

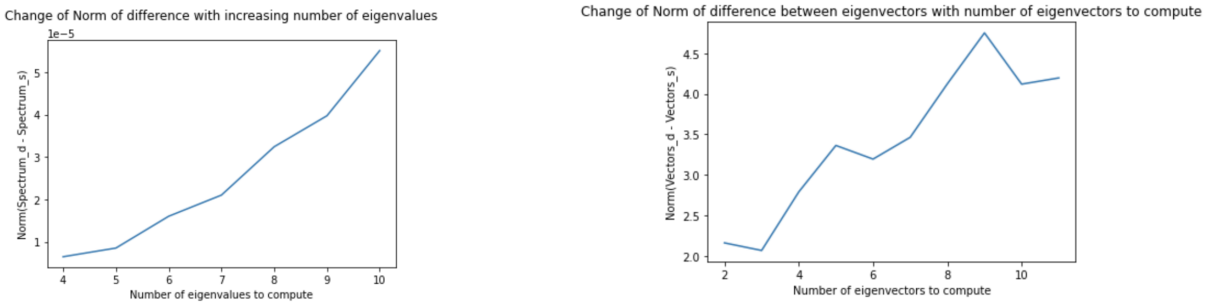


Figure 3.18: Error in eigenvalues and eigenvectors with increasing K

ture distribution over a rod in time. It satisfies the heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial \theta^2} \tag{3.3}$$

with boundary conditions

$$u(0, \theta) = g(\theta) \tag{3.4}$$

In order to solve the equation analytically, we need to employ separation of variables, a technique which enables us to rewrite the solution so that each variable appears on different side of the equation. This is a technique that assumes a solution of particular form, and is known as Fourier method:

$$u(t, \theta) = T(t)\Theta(\theta))$$

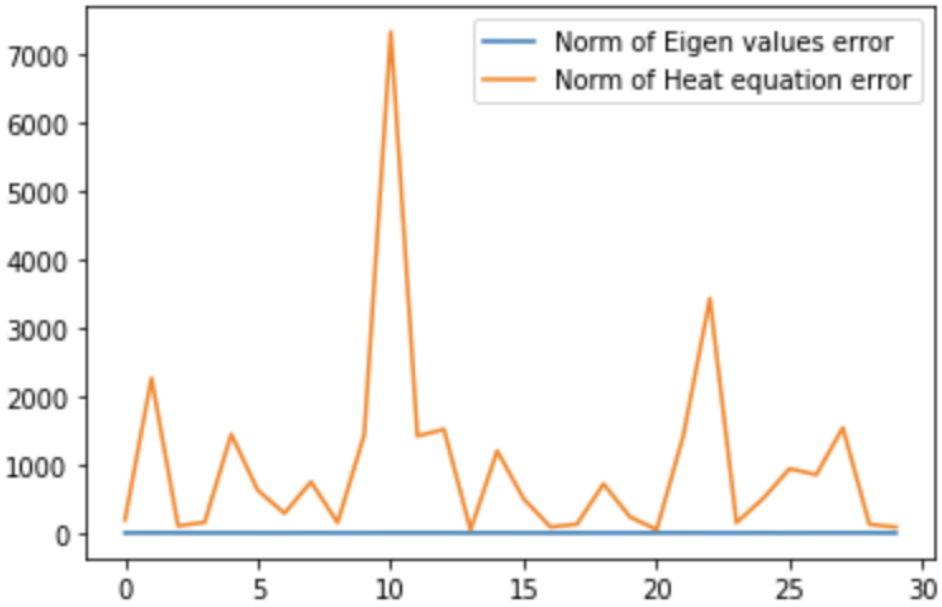


Figure 3.19: Norm of error of eigenvalues and heat equation solution obtained by SpectralNet, trained with the same parameters 30 times.

where $\Theta(T)$ and $T(t)$ are some functions that only depend on spatial and time variable, respectively. Substituting , the heat equation becomes:

$$T'(t)\Theta(\theta) = T(t)\Theta''(\theta).$$

If we rearrange the above equation, and get left hand side which only depends on t, and right hand side which only depends on $theta$, we know it is equal to some constant.

$$\frac{T'(t)}{T(t)} = \frac{\Theta''(\theta)}{\Theta(\theta)} = -\lambda$$

This gives us two equations:

$$\Theta'' + \lambda\Theta = 0$$

Where Θ is periodic function with period equal to 2π, and

$$T' + \lambda T = 0, t > 0$$

The solution of the latter problem is:

$$T(t) = e^{-k^2 t}$$

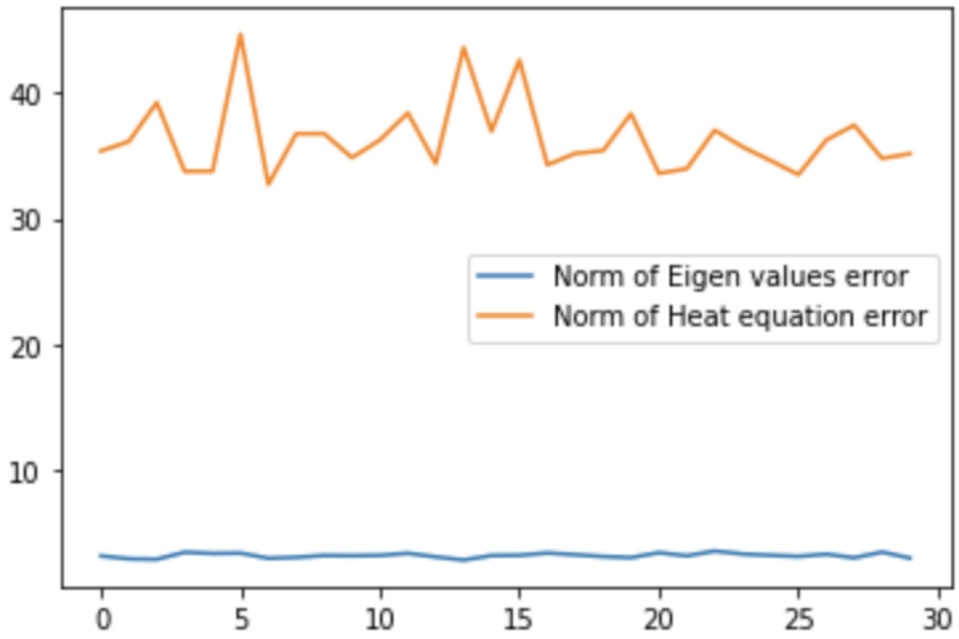


Figure 3.20: Norm of error of eigenvalues and heat equation solution obtained by SpectralNet, trained with the same parameters 30 times.

,

where $k^2 = \lambda$ and $k \in \mathbb{Z}$

$$\Theta(\theta) = \sum_{n=1}^{\infty} c_n e^{-\lambda_n} w_n(\theta)$$

where w_n is an eigenfunction corresponding to the eingevalues λ_n. The infinite sum reflects the fact that the heat equation has infinitely many real eigenvalues, $0 <= \lambda_1 <= \lambda_2 <= \lambda_3 ..., \lambda_n$. c_n are the coefficients such that:

$$u(0, \theta) = \sum_{n=1}^{\infty} c_n w_n(\theta)$$

The **eigenfunction** of a linear operator D defined on some function space is any non-zero function f in that space that, when acted upon by D, is only multiplied by some scaling factor called an eigenvalue. We know that $sin(wx)$ and $cos(wx)$ is a an eigenfunction of the operator $\frac{\partial^2 f}{\partial x^2}$. For example $\frac{\partial^2 f}{\partial x^2}(sin(wx) = -w^2 sin(wx)$., which proves that $sin(wx)$ and $-w^2$ is an eigenpair of this function.

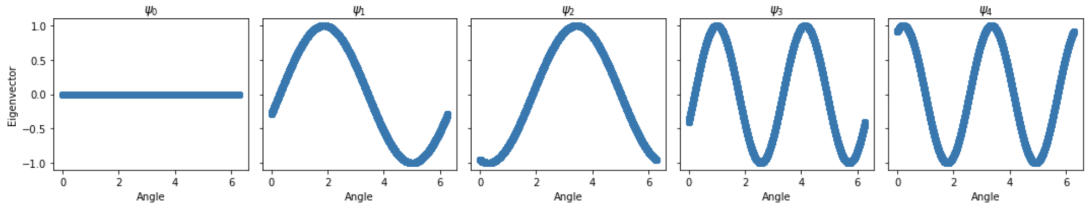


Figure 3.21: 5 Eigenvectors computed by datafold plotted against θ

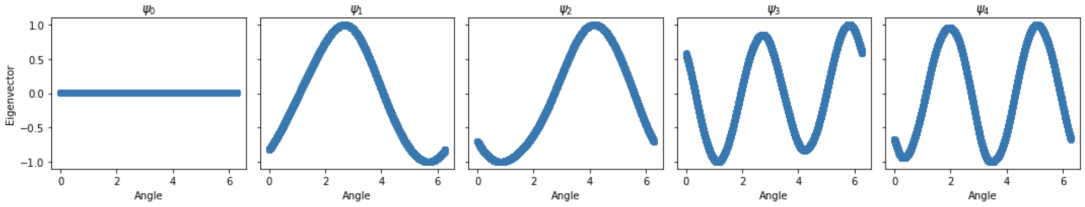


Figure 3.22: 5 Eigenvectors computed by spectralnet plotted against θ

Substituting the sines and cosines as eigenfunctions of the laplacian, we then have:

$$\Theta(\theta) = A_k cos(k\theta) + B_k sin(k\theta)$$

gives us solution to the first equation if $\lambda = k^2$ and $k \in \mathbb{N}$ for some constants A_k, B_k. Otherwise if $k = 0$, then $\Theta(\theta) = A_0$

The solution to the whole partial differential equation that satisfy the boundary conditions then becomes:

$$u_n(t, \theta) = e^{-k^2 t}(A_k cos(k\theta) + B_k sin(k\theta))$$

By linearity, the general is then given by superposition of all these solutions:

$$u(t, \theta) = A_0 + \sum_{k=1}^{\infty} e^{-k^2 t}(A_k cos(k\theta) + B_k sin(k\theta))$$

where $A_k and B_k$ are coefficients and the initial condition is satisfied if:

$$A_k = \frac{1}{\pi}\int_{-\pi}^{\pi} g(\theta) cos(k\theta)\, d\theta \quad \text{and} \quad B_k = \frac{1}{\pi}\int_{-\pi}^{\pi} g(\theta) sin(k\theta)\, d\theta \tag{3.5}$$

where $g(\theta) = u(0, \theta)$ is the initial condition, which is a function of heat distribution for the first timestamp, $t = 0$.

The smalles eigenvalue is 0, with corresponding constant eigenfunction of 1's. Then. as time goes to infinity, the temperature becomes diffuses uniformly over the whole domain and eventually reaches equilibrium and becomes a constant. Each term in the solution

contains negative exponential, and terms get faster and faster decaying since as n increases, λ_n increases quadratically. λ_2 represents the convergence rate towards the equilibrium point of the problem and A_0 determines the temperature it will converge to.

$$g(\theta) = \sum_{k=0}^{\inf} A_k cos(k\theta) + B_k sin(k\theta) = \frac{1}{2}A_0 + \sum_{k=1}^{\inf} A_k cos(k\theta) + B_k sin(k\theta)$$

The coefficients A_k and B_k are only dependent on the initial condition.
We take the initial condition to be 2π periodic function, for example:

$$u(0, \theta) = sin(\theta) + cos(2 * \theta)$$

And try to get the solution to the heat equation numerically, using the eigenvectors and eigenvalues obtained from our above-described methods. Apart from the eigenfunction and eigenvalue pairs, the coefficients that depend on the initial condition should also be computed.

To solve heat equation in python, we apply sine and cosine functions as true eigenfunctions, and compute the coefficients by solving a linear matrix equation with least-squares approach.

$$u(0, \theta) = \sum_{k=0}^{\inf} A_k cos(k\theta) + B_k sin(k\theta) = V \cdot c$$

where V are eigenfunctions of the laplacian and **c** is a vector of coefficients. **c** can be estimated with :

$$c = \underset{x}{\operatorname{argmin}} ||g(\theta) - Vx||.$$

Heat equation on uniformly sampled data

We are going to look at the solution to the heat equation over a domain of a circulad rod. We take our initial condition to be:

$$g(\theta) = sin(\theta) + cos(2 * \theta)$$

We compute first 5 eigenvectors with SpectralNet and Datafold.

We sample theta uniformly from a circle N=10000 times. We also sample time variable, t, uniformly 100 times, ranging from 0th to 10th seconds. By applying the initial condition of the heat to the sampled θ, we get the initial distribution of the heat. 3.23 shows the initial distribution of the heat, estimated with 3 different methods. Numerically from using approximationgs of eigenfunctions by SpectralNet and Datafold, and obtained by the exact solution (using sine and cosine functions with different frequencies).

To get the solution to the heat equation, we need to estimate Fourier coefficients. Approximating Fourier coefficients happens through least squares optimization using Numpy's linalg.lstsq method. We know, that our initial function can be approximated by sine and cosine functions weighted by some coefficients, and the least squares solution approximates them.

The infinite sum is then approximated by summing over number of computed eigenvalues and eigenvectors.

The results from analytical solution and two numerical solutions are plotted below. We see that the initial condition obtained from each solution is similar.

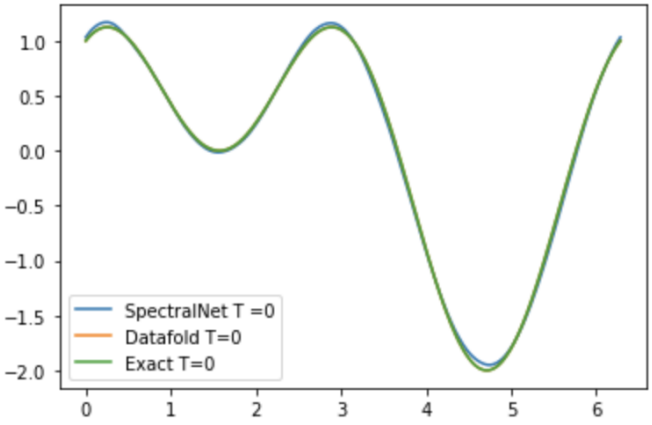


Figure 3.23: Initial distribution of heat obtaind from different method, plotted over the angle

But as we move away from initial timestamp, the difference between two solutions is apparent, as the first few eigenvectors and eigenvalues obtained from the neural network are not resembling the true eigenvalues and eigenfunctions exactly. As the time passes, the error then decreases exponentially for both numerical methods due to exponentially decaying relationship of the time t and value of the sum in the solution of the heat equation at time t. 3.24 we see how datafold has consistetly lower error compared to SpectralNet, it estimated the exact solution with $10^{-}-3$ precision. 3.25 shows us 3 solutions for the heat equation, plotted at different time stamps over the angle. As expected, Exact solution and Datafold are close to each other, however, the most important eigenvector and eigenvalue computed from SpectralNet deviates from the true one, which results in the difference in the solution for earlier timestamps, when diffusion first starts.

Heat equation on non-uniformly sampled data

The previous example resulted in neural network performance which was not comparable to standard industry solver's, however, it still managed to find eigenfunction subspace

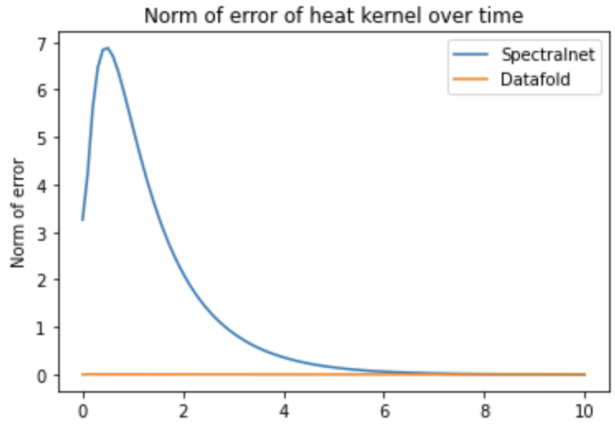


Figure 3.24: Distribution of norm of error over time

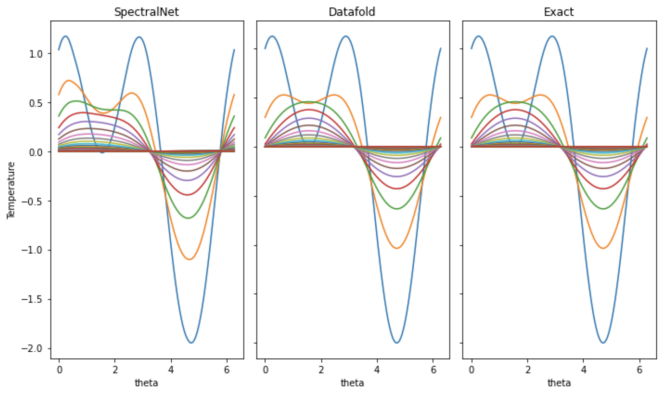


Figure 3.25: Distribution of the heat evolving over time, plotted over the angle θ

which was close to one spanned by true eigenfunctions. With this experiment, we try to identify whether the uniformity of data contributed to achieving low grassman distance between sets of eigenvectors and if in non-uniform case the network will be able to estimate eigenvalues with the same accuracy as previously.

We take our initial condition to be:

$$g(\theta) = sin(3 * \theta) + cos(\theta)$$

We compute first 7 eigenvectors with SpectralNet and Datafold.

In previous experiments, however, θ, the angle on a circular rod was sampled uniformly from 0 to 2π. Additionaly, another non-uniform samplig method was used to evaluate solution invariance to density of the problem.

$$\theta = 2 * \pi * (x^2), x \in [0,1)]$$

Evaluating the eigenvalues, as previously, still was accurate and the norm of difference was about $1.4e^{-05}$.

The eigenvectors, however, with the same parameter settings, are better estimated by SpectralNet than Datafold. Without adjusting the diffusion map parameters, the method fails to recognize the sine and cosine functions, however, spectral net was able to produce eigenvectors that better resemble the true eigenfunctions.

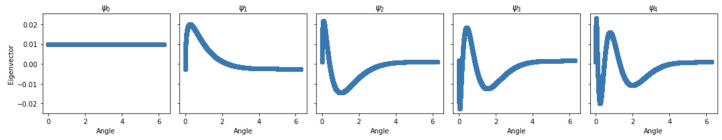


Figure 3.26: Eigenvectors computer by Datafold for non-uniform sampling

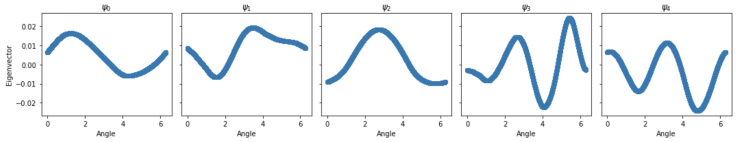
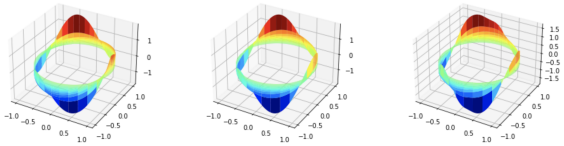


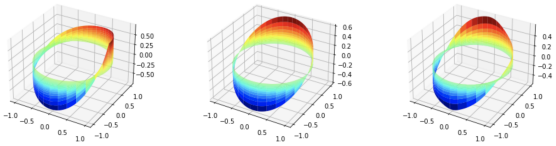
Figure 3.27: Eigenvectors computer by SpectralNet for non-uniform sampling

And therefore, the solution to the heat equation computed with SpectralNet's output is closer to the exact one compared to Datafold. Figures 3.28 illustrate the solutions over a domain for 3 different time stamps. 3.29 shows the error between obtained solutions for each timestamp, conversely to the uniform case, solution of Spectral Net approximates the solution better, because, without any parameters adjustments, the scale factor used in datafold skews the eigenvalues, while SpectralNet, although can not learn precise eigenfunctions and eigenvalues, can adjust itself to learn the approximation without externally changing it's configuration.

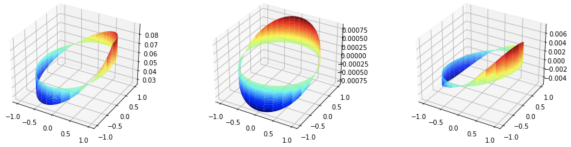
However, both datafold and spectralnet are far off from the true analytical solution, if we compute the solution this way. Datafold's hyperparameters can be adjusted to account for non-uniform data and still provide accurate representations of eigenfunctions, which, unfortunately is not yet achievable with non-linear neural network, even for the simper case of distributions.



(a) Initial heat distribution computed by Datafolt on the left, Analytically in the middle, and SpectralNet on the right



(b) Heat distribution at t=10 computed by Datafolt on the left, Analytically in the middle, and SpectralNet on the right



(c) Heat distribution at t=70 computed by Datafolt on the left, Analytically in the middle, and SpectralNet on the right

Figure 3.28: Datafold's numerical solution, Exact solution and SpectralNet's solution at time=0,10,70 from top to bottom

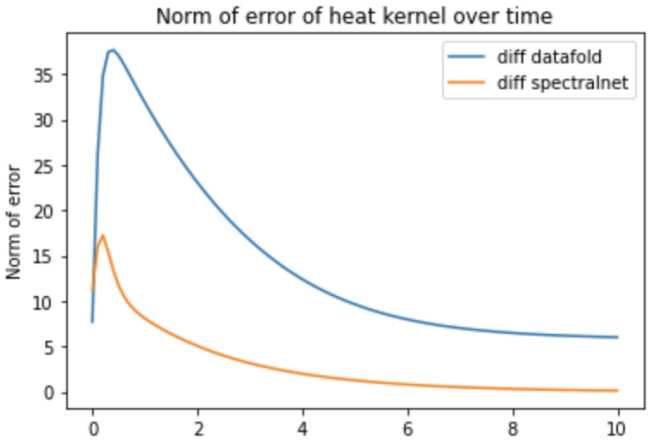


Figure 3.29: Distribution of norm of error over time for non-uniform sampling

4 Summary

4.1 Summary of the thesis

The aim of the thesis was to use and examine a novel approach towards solving eigenproblems, by employing complex non-linear functions for finding eigendecomposition of the Laplacian on a given domain. The potential usage of the neural network as an algorithm to solve eigenproblems was considered, as well as it's limitations and complications when it comes to learning the solution with a iterative, mini-batch based stochastic optimization procedure.

Apart from using a novel approach, a standard method for finding spectral properties of a matrix was chosen, as an alternative solution which was used as a subject of comparison for the novel approach. At first, eigenproblems were defined and the challenges of solving them have been introduced. Categorization of eigensolvers between direct and iterative types has been established, and several iterative solvers and the choice of those solvers for different kind of eigenproblems has been discussed. In this thesis, we have taken a closer look into two approaches that solve eigenproblems. For scalable and more straightforward solution, deep neural network based approach to eigenproblems was investigated and estimated. Potentially, neural networks are promising tool for finding approximations of eigenfunctions of the laplacian on a manifold, giving a more natural and compact solution to the problem than those provided by standard solvers with discretized methods and standard linear algebra operations. Apart from using it for standard clustering, it was applied to a test example, the heat equation on a circle and then compared with the solution obtained with another eigensolver. This standard approach to eigenproblems was performed by using diffusion map algorithm implemented in python package Datafold, which stands as a wrapper for state-of-the-art eigensolvers built in scipy, relying on Im-

plicitly Restarted Arnoldi Method (IRAM), which is the basis eigs routine implemented in ARPACK.

Section 1 introduces the problem of finding eigenvalues and eigenvectors of matrices and how different approaches have been used to solve it over the years. The importance of developing novel approaches in this field was discussed, as well as the challenges of computing eigenvalues and eigenproblems of a large matrix. In section 2 spectral clustering and diffusion maps were discussed, two dimensionality reduction methods that rely on eigenvectors of the normalized graph Laplacian. Both methods rely on eigensolvers and therefore have a limitation in terms of data size, since eigensolvers nowadays typically operate on the whole matrices and often are hard to parallelize or scale up. Diffusion map algorithm, implemented in software package Datafold, was used to estimate the eigenfunctions of the laplacian, and this section provided overview of the software and the eigensolvers it uses internally. Alternatively, to examine a newer, less well-known approach, a neural network, called SpectralNet was introduced, which primarily was designed to come up with spectral embeddings of the data for subsequent clustering, it's algorithm was briefly described, as well as other neural network approaches that perform clustering were compared. In section 3 Firstly, the architecture of neural network was presented, defining it's layers and number of neurons, as well as activation functions and custom layers specific to the problem at hand. The optimizer used for training the network was discussed and the loss function of the network was presented. Secondly, giving the unsupervised manner of the network learning and lack of adequate tools to evaluate the results, evaluation metrics were chosen to assess the quality of eigenvectors, given by the mapping of the data points to their spectral emdeddings. Apart from the orthogonality of the output, the Grassmann distance and projection error is measured as a way to estimate accuracy of the network and define how good the eigenfunction estimations really are compared to the true solution or the solution obtained by datafold. Moving forward, the problems to which SpectralNet was applied were presented. At first, the network was used for spectral clustering of data with non-linear decision boundary. The network, trained without any labeled data, managed to correctly identify the clusters in some cases, but failed in others, depending on the distribution of the data points between the clusters. More interestingly, the eigenvectors and eigenvalues of the graph Laplacian were estimated in order to later be used for a test example. The hyperparameter search for the parameters of the network was presented. Additionally, the choice of batch size parameter and it's importance was discussed in relation to the orthogonality of the output and general network performance. The sensitivity of neural network training was also considered to it's initialization. Additionally, the change in quality of eigenvectors was observed with added generalization capability to the network. Lastly, an application of the eigensolver was presented, applying the result to solve a partial differential equation numerically and compare the results with exact, analytical solution. The derivation of analytical solution was presented and the results from two eigensolvers were compared.

4.2 Discussion

Even though standard solvers available right now are fast and reliable, their limitations become apparent when it comes to performing eigendecomposition on large matrices. That's where neural network based approach offers its benefits, trading precision and accuracy for generalization ability and scalability.

Although Spectral Clustering and Diffusion maps can be applied to relatively large datasets as long as their similarity matrix is sparse, they cannot be used as a "black box" algorithm for clustering any kind of dataset. That is, without having to tune in parameters for each model, thinking about which similarity graph to use and how to choose its local neighborhood, these methods can be quite unstable. However, designing a network for spectral analysis could, after learning, serve as a universal tool for finding eigenfunctions of the Laplacian.

The results of our experiments show that although spectral clustering performance is better when using SpectralNet to get spectral embedding of the data for clustering, the eigenvector estimation is not as accurate or efficient when it comes to using them for other eigenproblems such as solving Partial Differential Equations, where precision of both eigenvalues and eigenvectors is important to get a solution that is similar to the true (analytically computed) one.

This lack of precision is largely due to instability of the network and stochastic nature of training. The choice of different mini-batches for each iteration and random initialization of the weights, gives a lot of variability to this process, which, in turn results in lack of accuracy of the output.

The evaluation of eigenvalues was relatively close to true eigenvalues, but this was largely due to the fact that the kernel matrix was estimated for the whole dataset, without mini-batches. If the procedure is changed to truly be fully scalable, then the approximation to heat diffusion would be even less accurate.

However, in some cases, for example when the data density of the problem is non-uniform, using a neural network could still offer it's benefits. Even though it is still not precise, with the same parameter settings, it manages to identify the approximations to the eigenfunctions of the Laplacian that resemble true sine and cosine functions, which, depending on the application of the solution (whether it asks for high precision or not), may alleviate the need for some of the hyperparameter search that is usually necessary for datafold.

4.3 Outlook

In general, neural network based eigensolver is an intriguing idea that needs further research. It is apparent, that the choice of architecture of this particular neural network did not affect the accuracy or the quality of eigenvectors. Further research could be employed to change the training procedure of this particular neural network, to be more stable than mini-batch training allows to be. Another approach could be changing the loss function of

the neural network in order to get more accurate eigenfunction estimation. There could be also some improvement in the type of layers used in the network, for example, in [10], a new type of layer was introduced that transformed symmetric positive semi definite matrix to another symmetric positive semi definite matrix, but in a different, more compact manifold, and then optimizing for the eigenvectors could become a more robust process, if, similarly to datafold, the layer would then learn to transform the input to its symmetric conjugate kernel. Another interesting approach that could also improve the results obtained from the network is using different initialization schemes, which could guide the function to learn a function towards producing orthogonal output.

Alternatively, a novel approach in Machine learning, weight agnostic neural networks could be employed to tackle the same problem, by searching for an optimal neural network architecture that can achieve a certain task with just random weights, to overcome the problem of training difficulty for this particular problem. Alternatively, as already mentioned, other reinforcement learning ideas have been brought up that solve the eigenproblem as a multi-agent game [11].

Substituting the standard solvers with neural network based approaches seem to be an idea too far in the future, however, as in a lot of numerical methods, using less accurate preconditioners for other solvers is a standard approach, and this could be another interesting research idea. As described in [15], machine learning could greatly benefit numerical linear algebra by aiding its iterative procedure with efficient deep learning based preconditioners. Preconditioners are an extra step added to numerical methods to accelerate their convergence. In fact, a lot of standard eigensolvers, that serve large scale eigenvalue computations, are solving preconditioned system via iterative methods. In a lot of cases, the convergence rate of those methods is highly dependent on the good initial starting point, and neural network could be utilised to provide starting points for those solvers, such as Newton's method or Jacobi-Davidson method described in [36], which could easily be preconditioned.

Bibliography

[1] Shrirang Abhyankar, Jed Brown, Emil M Constantinescu, Debojyoti Ghosh, Barry F Smith, and Hong Zhang. Petsc/ts: A modern scalable ode/dae solver library. *arXiv preprint arXiv:1806.01437*, 2018.

[2] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 585–591, Cambridge, MA, USA, 2001. MIT Press.

[3] Abdelkarim Ben Ayed, Mohamed Ben Halima, and Adel Alimi. Adaptive fuzzy exponent cluster ensemble system based feature selection and spectral clustering. pages 1–6, 07 2017.

[4] Yoshua Bengio, Jean-François Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, page 177–184, Cambridge, MA, USA, 2003. MIT Press.

[5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[6] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. Special Issue: Diffusion Maps and Wavelets.

[7] K.Ch. Das. The laplacian spectrum of a graph. *Computers and Mathematics with Applications*, 48(5):715–724, 2004.

[8] Chris Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst Simon. A min-max cut algorithm for graph partitioning and data clustering. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 107-114, 11 2001.

[9] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystr??m method. *IEEE transactions on pattern analysis and machine intelligence*, 26:214–25, 03 2004.

[10] Zhi Gao, Yuwei Wu, Xingyuan Bu, Tan Yu, Junsong Yuan, and Yunde Jia. Learning a robust representation via a deep network on symmetric positive definite manifolds. *Pattern Recognition*, 92:1–12, 2019.

[11] Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. Eigengame: {PCA} as a nash equilibrium. In *International Conference on Learning Representations*, 2021.

[12] Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. Eigengame unloaded: When playing games is better than optimizing, 2021.

[13] Gene H. Golub and Henk A. van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1):35–65, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.

[14] Yanan Guo, Xiaoqun Cao, Bainian Liu, and Mei Gao. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences*, 10(17), 2020.

[15] Markus Götz and Hartwig Anzt. Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pages 49–56, 2018.

[16] Jihun Ham, Daniel Lee, Sebastian Mika, and Bernhard Schölkopf. A kernel view of the dimensionality reduction of manifolds. 07 2004.

[17] Denis Hamad and Philippe Biela. Introduction to spectral clustering. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–6, 2008.

[18] Yufei Han and Maurizio Filippone. Mini-batch spectral clustering, 2016.

[19] Magnus R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–435, 1952.

[20] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering, 2017.

[21] Daniel Lehmberg, Felix Dietrich, Gerta Köster, and Hans-Joachim Bungartz. datafold: data-driven models for point clouds and time series on manifolds. *Journal of Open Source Software*, 5(51):2283, 2020.

[22] R. Lehoucq, D. Sorensen, and C. Yang. Arpack users' guide - solution of large-scale eigenvalue problems with implicitly restarted arnoldi methods. In *Software, environments, tools*, 1998.

[23] Henry Li, Ofir Lindenbaum, Xiuyuan Cheng, and Alexander Cloninger. Variational diffusion autoencoders with random walk sampling, 2020.

[24] G. Mathew and V. Reddy. Orthogonal eigensubspace estimation using neural networks. *IEEE Trans. Signal Process.*, 42:1803–1811, 1994.

[25] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 849–856, Cambridge, MA, USA, 2001. MIT Press.

[26] J Porte, Ben Herbst, Willy Hereman, and Stéfan van der Walt. An introduction to diffusion maps. 11 2008.

[27] Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks, 2018.

[28] Hiroyuki Shinnou and Minoru Sasaki. Spectral clustering for a large data set by reducing the similarity matrix size. 01 2008.

[29] S. Suryanarayana, D. Rao, and D. Swamy. A survey : Spectral clustering applications and its enhancements. 2015.

[30] Ulrike von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, 36(2), Apr 2008.

[31] Jinghua Wang and Jianmin Jiang. Sa-net: A deep spectral analysis network for image clustering, 2020.

[32] Xinchao Wang, Zhu Li, and Dacheng Tao. Subspaces indexing model on grassmann manifold for image search. *Image Processing, IEEE Transactions on*, 20:2627 – 2635, 10 2011.

[33] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation, 2016.

[34] Zhang Yi, Yan Fu, and Hua Jin Tang. Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix. *Computers and Mathematics with Applications*, 47(8):1155–1164, 2004.

[35] Li Zhou, Lihao Yan, Mark A. Caprio, Weiguo Gao, and Chao Yang. Solving the k-sparse eigenvalue problem with reinforcement learning, 2020.

[36] Yunkai Zhou. Eigenvalue computation from the optimization perspective: On jacobi-davidson, iigd, rqi, and newton updates. *arXiv: Numerical Analysis*, 2004.