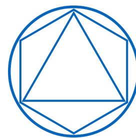
Technische Universität München

Department of Mathematics



Master’s Thesis in Mathematics in Data Science

Properties of Linear Operators Related to Gaussian Processes

Fan Xu

Supervisor: Prof. Dr. Hans-Joachim Bungartz
Advisor: Dr. Felix Dietrich
Submission Date: 01.05.2022

I confirm that this master's thesis in Mathematics in Data Science is my own work and I have documented all sources and materials used.

Munich, 01.05.2022

Acknowledgements

I would like to thank my supervisor Prof. Dr. Hans-Joachim Bungartz for his kind willingness to supervise me and my advisor Dr. Felix Dietrich for his constructive advice. The most sincere gratitude gives to my parents for their long-time encouragement and solid financial support even though their small business was and is suffering a lot due to Covid-19 pandemic.

Abstract

It is interesting that neural networks can be turned into Gaussian processes in the limit of infinite neurons in hidden layers, from which we convert neural networks into familiar probabilistic models that are already well studied. Gaussian processes are defined by kernel functions, which are actually much related to linear operators. By the virtue of rich and sophisticated theories that have been well developed for linear operators, we can study neural networks through linear operators, which brings the theoretical richness of linear operators to neural networks. Through the study of spectral properties of linear operators that relate to Gaussian processes, the evolving geometric structure between original data points and the outputs of networks is further understood.

In this thesis, we first illustrate background knowledge pertaining to linear operators, kernel functions, nonlinear dimensionality reduction methods, Gaussian Processes for regression and classification, and the equivalence of neural networks with Gaussian processes. After that, we experimentally explore the influence of altering the convolutional network depths and differing the number of input images on eigenanalysis of kernel matrices induced by convolutional neural networks as Gaussian processes (CNN-GPs) and residual convolutional neural networks as Gaussian processes (ResCNN-GPs). We observe that eigenvectors are very similar for networks with different depths while eigenvalues are greater for networks with more layers on MNIST dataset and skip connection does not significantly affect the geometrical structure of eigenvectors as well. We also investigate the classification accuracy with respect to network depths of CNN-GPs and ResCNN-GPs, showing that both CNN-GPs and ResCNN-GPs with 7 layers have the best accuracy. When analyzing classification accuracy of CNN-GPs we conclude that larger training kernel and smaller testing kernel produce a higher accuracy.

Contents

1 Introduction 1

2 Related Work 3
2.1 Linear Operators . . . 3
2.1.1 Eigenfunctions and Eigenvalues of Linear Operators . . . 4
2.1.2 Kernels of Linear Operators . . . 5
2.2 Kernels . . . 5
2.2.1 Kernel Functions . . . 5
2.2.2 Eigenfunction Analysis of Kernels . . . 10
2.3 Nonlinear Dimensionality Reduction . . . 12
2.4 Gaussian Processes . . . 14
2.4.1 Covariance Functions . . . 16
2.4.2 Gaussian Processes for Regression . . . 18
2.4.3 Gaussian Processes for Classification . . . 21
2.5 Neural Networks . . . 24
2.5.1 Neural Networks as Gaussian Processes . . . 25
2.5.2 Kernels of Neural Networks . . . 29

3 Properties of Linear Operators Related to Gaussian Processes 32
3.1 Hilbert-Schmidt Integral Operator . . . 32
3.1.1 The Integral Operator for Kernels . . . 32
3.1.2 Properties of the Integral Operator . . . 34
3.1.3 An Example Based on Radial Basis Function . . . 36
3.2 Networks with Different Architectures . . . 36
3.2.1 Eigen-analysis of Kernel Matrices . . . 37
3.2.2 Performance on Classification . . . 54
3.3 Networks with Different Number of Input Images . . . 59
3.3.1 Eigen-analysis of Kernel Matrices . . . 59
3.3.2 Performance on Classification . . . 62

4 Conclusion and Future Work 64

Bibliography 65

1 Introduction

With the booming development of computer science nowadays, many aspects of our lives have been changed significantly or are under rapid evolution. Of many research fields, machine learning is one of the intriguing fields that attract much attention from researchers recently. Supervised learning utilizes data with continuous or discontinuous labels and unsupervised learning uses data without labels. For labeled data, regression with respect to continuous function outputs and classification with discontinuous targets are two principal problems in supervised learning. For unlabeled data, our main interests lay in discovering patterns or low dimensional embedding of training data. Various algorithms have been developed to deal with different types of problems and datasets. For example, we can use decision trees for classification and least squares method for regression, while K-means is used to cluster the unlabeled data, and principal component analysis is used to reduce the dimensionality of linear data in order to find the submanifold of such data. Moreover, deep learning is especially fascinating to scientists in recent years. With the advantages of neural networks, there is no need to know the explicit form of functions that fit the true latent function, we only need to choose certain architectures suitable for processing our training data with some initialization setups, proper loss functions, and appropriate gradient descent methods, then a very good performance on test data with high accuracy and low loss can be achieved.

Neural networks are machine learning models that well solve problems regarding classification and regression. Nevertheless, classic neural networks tend to overfit datasets due to the huge amount of parameters within the network. The over-parametrization allows for convergence to global minima, where the training error is zero or nearly zero [2]. Intuitively, when the amount of neurons goes to infinity we will have a network with infinitely many parameters. However, Neal [21] has proven that the one-layered neural networks with randomly distributed weights tend to a Gaussian process over the input data in the limit of infinite neurons in hidden layers. Furthermore, Williams [32] showed that predictions can be made efficiently using infinitely many hidden neurons and it was easier to compute the infinite networks than finite ones through the concrete mathematical forms of the covariance functions. Gaussian processes are probabilistic methods that are widely used in regression and classification problems. Gaussian process regression uses posterior probability to predict distributions of new data points. In contrast, Gaussian process classification uses logistic regression function in generative models, while for discriminative models, Sigmoid function is used for binary classification and the Softmax function is used for multi-class classification.

When we consider neural networks as Gaussian processes, the problem will be much easier to analyze due to the sophisticated theories that have been developed. A Gaussian process is determined by its mean and covariance function, of which the covariance function is the most crucial. To utilize the rich theories of linear operators, we define a linear operator using covariance functions — the Hilbert-Schmidt integral operator. Once the operator is defined, we therefore define eigenfunctions and eigenvalues of a covariance function. And Mercer's theorem gives us an explicit representation of covari-

ance function through eigenfunctions and eigenvalues, which helps a lot when we intend to explore the geometric structure of data points.

In Section 2, we will demonstrate the necessary knowledge and essential work concerning linear operators, kernels, nonlinear dimensionality reduction, and Gaussian Processes as well as the connection between neural networks and Gaussian processes. In Section 3, we will mainly analyze the properties of the integral operator on kernel matrices induced by convolutional neural networks and investigate the influence of kernel matrices with differing network architectures and varying kernel sizes on eigenvectors and eigenvalues. Related performance on classification is conducted as well in order to observe how varying kernels affect classification accuracy.

2 Related Work

In order to have a comprehensive understanding of the contents of this thesis, we first present the fundamental knowledge stemmed from related work done by other researchers. We start with linear operators in linear algebra, based on the expositions in [1, 9].

2.1 Linear Operators

Operators are usually mappings that map the elements from one space to another space. A operator $\mathcal{L} : U \to V$ is called a linear operator if it has the following two properties:

(i) Additivity: $\mathcal{L}(x + y) = \mathcal{L}(x) + \mathcal{L}(y)$ for all x and $y \in U$;

(ii) Homogeneity: $\mathcal{L}(cx) = c\mathcal{L}(x)$ for all $x \in U$ and all constant $c \in \mathbb{R}$,

where U and V are vector spaces. With the above definition, it follows

$$\mathcal{L}(ax + by) = a\mathcal{L}(x) + b\mathcal{L}(y) \tag{2.1}$$

for any constant $a, b \in \mathbb{R}$ and variable $x, y \in U$. The composition of two linear operators $\mathcal{L}_1 : U \to V$ and $\mathcal{L}_2 : V \to W$ is a new linear operator $\mathcal{L}_2 \circ \mathcal{L}_1 : U \to W$:

$$\begin{aligned} \mathcal{L}_2 \circ \mathcal{L}_1(ax + by) &= \mathcal{L}_2 \circ \mathcal{L}_1(ax) + \mathcal{L}_2 \circ \mathcal{L}_1(by) \\ &= a\mathcal{L}_2 \circ \mathcal{L}_1(x) + b\mathcal{L}_2 \circ \mathcal{L}_1(y). \end{aligned} \tag{2.2}$$

The order of linear operators is comparatively essential and operators are not commutative generally.

The inverse of a linear operator $\mathcal{L}$ is defined as $\mathcal{L}^{-1}$, which satisfies $\mathcal{L} \circ \mathcal{L}^{-1} = \mathcal{I}$, where $\mathcal{I}$ is the identity operator (it is also a linear operator since we get $\mathcal{I}(x) = x$). It is worth noting that not every operator has an inverse.

There are many linear operators. A linear function $f : x \to cx$ from $\mathbb{R}$ to $\mathbb{R}$ is the simplest linear operator. Moreover, differentiation and integration are two commonly used linear operators in Mathematics as well. According to Equation 2.1, we have

$$\frac{d(ax + by)}{dt} = a\frac{dx}{dt} + b\frac{dy}{dt} \tag{2.3}$$

for differentiation; while for integration, we have:

$$\int_{\Omega} (ax + by)dt = a \int_{\Omega} xdt + b \int_{\Omega} ydt, \tag{2.4}$$

where Ω is the domain of integral, a and b are constants, and x and y the functions of t, respectively. We will later apply the Hilbert-Schmidt integral operator on covariance matrices (also called kernel matrices) of Gaussian processes deduced from convolutional neural networks in Section 3, such that we can investigate the change of eigenfunctions and corresponding eigenvalues of kernel matrices.

2.1.1 Eigenfunctions and Eigenvalues of Linear Operators

For finite dimensional vector spaces U and V with a basis defined on each space, we then can represent any linear operator from U to V by a matrix. For instance, if $\boldsymbol{x} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can define a linear operator $\mathcal{L}$ which maps $\mathbb{R}^n \to \mathbb{R}^m$ by

$$\mathcal{L}(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x}, \tag{2.5}$$

from which the linear operator $\mathcal{L}$ is represented by matrix $\mathbf{A}$. For more general cases, suppose we have a basis $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_n\}$ for U and a basis $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_m\}$ for V, then a vector $\boldsymbol{u} \in U$ can be uniquely represented by:

$$\boldsymbol{u} = a_1\boldsymbol{u}_1 + \cdots + a_n\boldsymbol{u}_n. \tag{2.6}$$

For a linear operator $\mathcal{L} : U \to V$, we obtain:

$$\begin{aligned} \mathcal{L}(\boldsymbol{u}) &= \mathcal{L}(a_1\boldsymbol{u}_1 + \cdots + a_n\boldsymbol{u}_n) \\ &= a_1\mathcal{L}(\boldsymbol{u}_1) + \cdots + a_n\mathcal{L}(\boldsymbol{u}_n), \end{aligned} \tag{2.7}$$

where $\mathcal{L}(\boldsymbol{u}_i) = b_{1i}\boldsymbol{v}_1 + \cdots + b_{mi}\boldsymbol{v}_m$ for $i = 1, \dots, n$. Therefore, we can rewrite Equation 2.7 through

$$\mathcal{L}(\boldsymbol{u}) = (\mathbf{B}\boldsymbol{a})^T\mathbf{V}, \tag{2.8}$$

where $\boldsymbol{a} = (a_1, \dots, a_n)^T$, $\mathbf{V} = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_m)$ and

$$\mathbf{B} = \begin{pmatrix} b_{11} & \cdots & b_{1i} & \cdots & b_{1n} \\ b_{21} & \cdots & b_{2i} & \cdots & b_{2n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mi} & \cdots & b_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}. \tag{2.9}$$

Here we just transform one vector space to another vector space through a linear transformation $\mathbf{B}\boldsymbol{a}$, where $\mathbf{B}$ is the matrix of $\mathcal{L}$. Note that the matrix for a linear operator is not unique, as we can choose various bases for a vector space.

As shown above, there is a connection between linear operators and matrices, we then can analyze the eigenvectors and eigenvalues for a linear operator through its matrix.

Given a linear operator $\mathcal{L} : U \to U$, a nonzero vector $\boldsymbol{x} \in U$ and a scalar λ are called eigenvector and its corresponding eigenvalue, respectively, if we have

$$\mathcal{L}(\boldsymbol{x}) = \lambda\boldsymbol{x}. \tag{2.10}$$

Since the linear operator $\mathcal{L}$ can be represent by a square matrix $\mathbf{A}$, Equation 2.10 is rewritten as:

$$\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}. \tag{2.11}$$

If any eigenvalue $\lambda = 0$, then the matrix $\mathbf{A}$ is not invertible, which means that matrix $\mathbf{A}$ does not have a full rank and its determinant equals 0. All the vectors that satisfy $\mathbf{A}\boldsymbol{x} = \mathbf{0}$ for $\forall \boldsymbol{x} \in U$ form the null space of matrix $\mathbf{A}$.

A very important property concerning eigenvectors and eigenvalues is that they are invariant of square matrices with changing of basis, as they are defined in terms of linear operators, not in terms of matrices. Suppose we have two matrices $\mathbf{A}$ and $\mathbf{B}$ that represent linear operator $\mathcal{L}$ under two different bases, then $\mathbf{A}$ is conjugated to $\mathbf{B}$ thus $\mathbf{B} = \mathbf{Q}^{-1}\mathbf{AQ}$ holds, where $\mathbf{Q}$ is the transition matrix between two bases. Clearly, $\mathbf{A}$ and $\mathbf{B}$ have identical eigenvalues, and their eigenvector spaces are isomorphic to each other.

2.1.2 Kernels of Linear Operators

If $\mathcal{L} : U \to V$ is a linear operator, then the kernel of the operator is defined as

$$\ker(\mathcal{L}) = \{\boldsymbol{x} \in U : \mathcal{L}(\boldsymbol{x}) = \mathbf{0}\}, \tag{2.12}$$

where $\mathbf{0}$ is the zero vector. That is to say, the kernel of a linear operator is a subspace of U consisting of vectors whose images are $\mathbf{0}$ in V. The null space of a matrix and the kernel of a linear operator are the same if linear operators $\mathcal{L}$ can be represented by a matrix. As a matter of fact, neural networks can be regarded as operators: with activation being linear functions the network is a linear operator, while the network is not a linear operator if the activation function is nonlinear. Hence finding the kernel of a linear neural network will be equivalent to finding all the training points in space U that are outputted as $\mathbf{0}$'s in V.

The kernel of a Gaussian process is the fundamental solution of the equation $\boldsymbol{u}_t = \mathcal{L}(\boldsymbol{u}), \boldsymbol{u} \in U$, which contains functions that satisfy the condition $\boldsymbol{u}_t = \mathcal{L}(\boldsymbol{u}) = \mathbf{0}$. And we can see that $\boldsymbol{u}$ is the kernel of operator $\frac{\partial}{\partial t} - \mathcal{L}$ due to equation $(\frac{\partial}{\partial t} - \mathcal{L})\boldsymbol{u} = \mathbf{0}$.

2.2 Kernels

Kernel functions $k(\boldsymbol{x}, \boldsymbol{x}')$ measure the similarity between two vectors $\boldsymbol{x}$, $\boldsymbol{x}'$. The greater the value of a kernel function is, the more similar the two vectors are. The reason we use kernel functions is that we do not need to specify an explicit form for a feature map $\boldsymbol{\psi}(\boldsymbol{x})$, while classic methods require data to be transformed to an explicit feature vector, which may be difficult for some problems. There are plenty of kernels that are used according to the concrete problems, and we will introduce some of them in more detail. The main ideas of kernels are from [4, 20, 25].

2.2.1 Kernel Functions

A kernel is defined as a real-valued function $k(\boldsymbol{x}, \boldsymbol{x}')$ of two vectors $\boldsymbol{x}$, $\boldsymbol{x}' \in \mathcal{X} \subset \mathbb{R}^n$. If a kernel function $k(\cdot, \cdot)$ satisfies the following two properties:

- The kernel is symmetric, namely $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$;
- The kernel is non-negative, that is to say $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$,

then it is called a Mercer kernel. A kernel can be written to the form $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{y}) \rangle$, where $\boldsymbol{\psi}(\cdot)$ maps $\boldsymbol{x}, \boldsymbol{y}$ into a higher dimensional space.

To judge whether a kernel is valid or not, a necessary and sufficient condition is that the Gram matrix $\mathbf{K}$ (also called kernel matrix) should be symmetric for all nonempty subsets of points $\{\boldsymbol{x}_i, i = 1, \ldots, n\}$ and positive semidefinite (PSD), i.e. for any vector $\boldsymbol{a}$ it has $\boldsymbol{a}^T \mathbf{K} \boldsymbol{a} \geq 0$.

Many kernels are available [25]. We will introduce some basic kernels from the simplest form to a more complicated form. The simplest kernel is the constant kernel, which has the form

$$k(\boldsymbol{x}, \boldsymbol{x}') = c, \tag{2.13}$$

where $c \in \mathbb{R}$. Actually, a constant kernel does not really measure the similarity between two vectors, so it is not really used alone in practice, instead of as a scaling factor combined with other kernels. Another simple kernel is the white kernel, which gives a nonzero noise level for two identical vectors and returns 0 for two varying vectors. We thus have a diagonal Gram matrix for the white kernel. In addition to the constant kernel or white kernel, linear kernel (also called dot-product kernel) is often used when data is linearly separable and the number of features is large, which is denoted by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \boldsymbol{x}^T \boldsymbol{x}' + c, \tag{2.14}$$

where σ_f is the vertical variation coefficient and c is a constant in $\mathbb{R}$. The polynomial kernel is commonly used in Support Vector Machines (SVM), which allows our model to learn nonlinearity

$$k(\boldsymbol{x}, \boldsymbol{x}') = (a\boldsymbol{x}^T \boldsymbol{x}' + b)^c, \tag{2.15}$$

where a, b and c are constants. If $c = 0$, Equation 2.15 will be a constant kernel; if $c = 1$ Equation 2.15 is a linear kernel, which makes polynomial outperform linear kernels due to its versatility. The most frequently used kernels is Radial Basis Function (RBF, also named Gaussian kernel), which has the form:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{d(\boldsymbol{x}, \boldsymbol{x}')^2}{2l^2}\right), \tag{2.16}$$

where σ_f is the vertical variation coefficient, and l is the length scale and $d(\cdot, \cdot)$ the Euclidean distance. The RBF is isotropic, which means that the scaling parameter $\gamma = \frac{1}{2l^2}$ scales the same amount in all dimensions, and it maps an finite dimensional vector space into an infinite dimensional vector space. In Section 2.3.2, we will explain more about the influence of these parameters on RBF. We should note that RBF is a stationary kernel, which is invariant to the translation, i.e. $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} + \boldsymbol{c}, \boldsymbol{x}' + \boldsymbol{c})$, due to the Euclidean distance used in the exponential. It is also infinitely smooth since it has an analytical derivative for any order. RBF has a good performance when applied to linearly inseparable data, but it fails when our latent function is periodic. Nonetheless, periodic kernel comes to the rescue. The periodic kernel is very useful for modeling periodic functions:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi d(\boldsymbol{x}, \boldsymbol{x}')}{p}\right)}{l^2}\right), \tag{2.17}$$

where $p > 0$ and $l > 0$. It should be known that periodic kernel is also stationary. Rational Quadratic kernel can be seen as an infinite sum of RBF with different l:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \frac{d(\boldsymbol{x}, \boldsymbol{x}')^2}{2al^2}\right)^{-a}, a > 0 \text{ and } l > 0, \tag{2.18}$$

which will result in a smooth prior on functions. When $a \to \infty$ the rational quadratic kernel converges to RBF. Furthermore, Matern kernel is a more general kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{\Gamma(v)2^{v-1}} \left(\frac{\sqrt{2v}}{l} d(\boldsymbol{x}, \boldsymbol{x}')\right)^v K_v \left(\frac{\sqrt{2v}}{l} d(\boldsymbol{x}, \boldsymbol{x}')\right), \tag{2.19}$$

where $l > 0$, $d(\cdot, \cdot)$ is Euclidean distance, $K_v(\cdot)$ is a modified Bessel function of order v and $\Gamma(\cdot)$ the gamma function. When $v \to \infty$, the Matern kernel converges to RBF.

Figure 2.1 gives us intuitive impressions of corresponding kernel functions. The values of parameters are shown in Table 2.1.

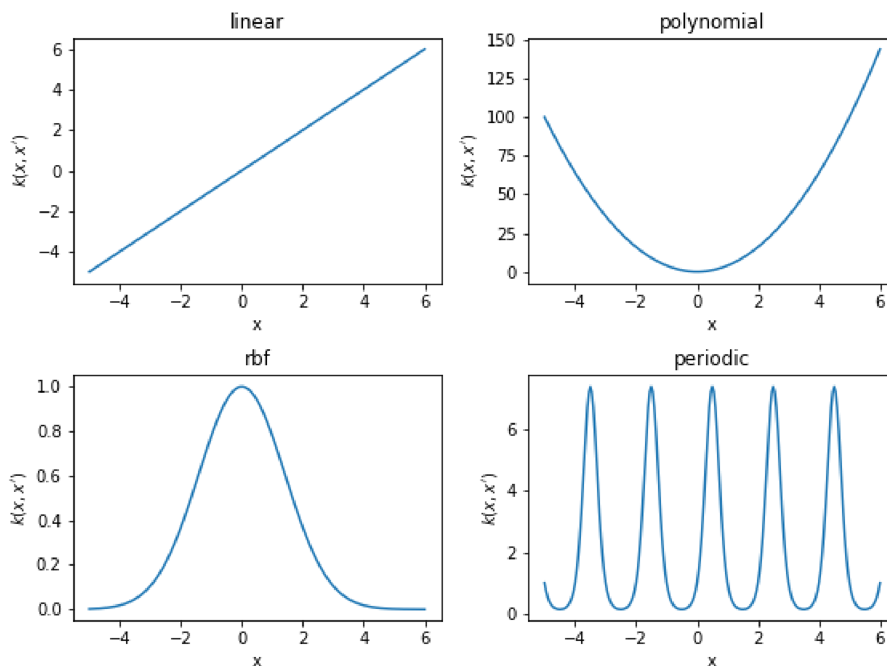


Figure 2.1: Graphs of different kernel functions with 1D inputs x. The top left is obviously linear kernel; the top right graph is the polynomial kernel; the bottom left curve is the curve of RBF, which has a shape of the probability density function of the normal distribution; the bottom right one is the plot of the periodic kernel, which has a notable periodicity.

We can use existing kernels to construct new kernels that are suitable for specific problems. Some constructing rules are as follows:

Kernels	Parameters and their values
Linear	$\sigma_f = 1, c = 0, x' = 1$
Polynomial	$a = 1, b = 0, c = 2, x' = 2$
RBF	$\sigma_f = 1, l = 2, x' = 0$
Periodic	$\sigma_f = 1, l = 1, p = 1, x' = 0$

Table 2.1: Kernels and their parameter values used in Figure 2.1.

- Sum kernels:
$$k_{sum}(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{2.20}$$
- Production kernels:
$$k_{prod}(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') * k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{2.21}$$
- Exponential kernels:
$$k_{exp}(\boldsymbol{x}, \boldsymbol{x}') = [k_1(\boldsymbol{x}, \boldsymbol{x}')]^p \tag{2.22}$$

For more details of constructing new kernels, we refer to [4] and [25].

Kernels are widely applied in Machine Learning. And the most familiar applications in Machine Learning are Kernel Principal Component Analysis (kPCA) [28] and Support Vector Machines (SVMs) [6].

SVMs are supervised learning methods that can be used for regression and classification. For simplicity we just talk SVMs for binary classification on dataset $\mathcal{X} = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, n\}$ where $\boldsymbol{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$. The SVM aims to maximize the width of margin between two categorical data points:

$$\begin{aligned} &\min \frac{1}{2} |\boldsymbol{w}|^2, \\ &\text{s.t. } y_i(\boldsymbol{w}\boldsymbol{x}_i + w_0) \geq 1, \text{for all } i = 1, \ldots, n, \end{aligned} \tag{2.23}$$

such that new points can be classified to one category based on

$$\text{sgn}(\boldsymbol{w}\boldsymbol{x}_* + w_0) = \text{sgn}(\sum_{i=1}^{n} \lambda_i y_i (\boldsymbol{x}_i \boldsymbol{x}_*) + w_0), \tag{2.24}$$

where $\boldsymbol{w}$ is the weighting vector, w_0 is the bias term, and $\boldsymbol{x}_*$ is a new data point that we want to classify. By the virtue of kernels, we can perform classification on linearly inseparable data through SVMs by replacing $\boldsymbol{x}_i \boldsymbol{x}_*$ by a kernel function $k(\boldsymbol{x}, \boldsymbol{x}_*)$. The representer theorem [26] ensures a solution for the minimization problem of SVM, which is a linear combination of kernels centered on data points. Figure 2.2 shows the binary classification of SVM for 500 data points generated by **make_blobs()** in sklearn, from which we see a good classification result with nonlinear boundary and support vectors are marked with green dots. A so-called soft margin SVM is also used for linearly inseparable binary classification when we have some outliers for each class. Although the algorithm allows misclassification, the misclassification error should be minimized.

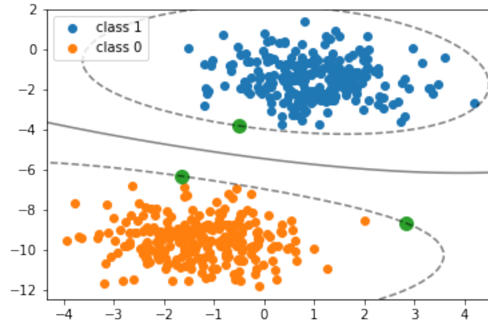


Figure 2.2: Binary classification using SVM with RBF. Yellow dots are points labeled with 0 and blue dots are points labeled with 1. Dashed lines are boundaries for different classes. Green dots are support vectors on boundary.

Classic PCA is applied to reduce the dimensionality of linear data if we discover that data is embedded in a submanifold, but kPCA is utilized for the dimensionality reduction of nonlinear data. Kernel PCA first maps linearly inseparable data in $\mathbb{R}^n$ into a higher dimensional space $\mathbb{R}^N, N > n$, then it projects the higher dimensional data back to a low dimensional space such that the projected data can be linearly separable. Figure 2.3 shows the advantage of kPCA over PCA on linearly inseparable data from **make_moons()** in Python library sklearn. It turns out that the classic PCA still reserves linear inseparability of our data, but the kPCA gives us transformed data that can be perfectly separated by a straight line, for which we use the RBF kernel.

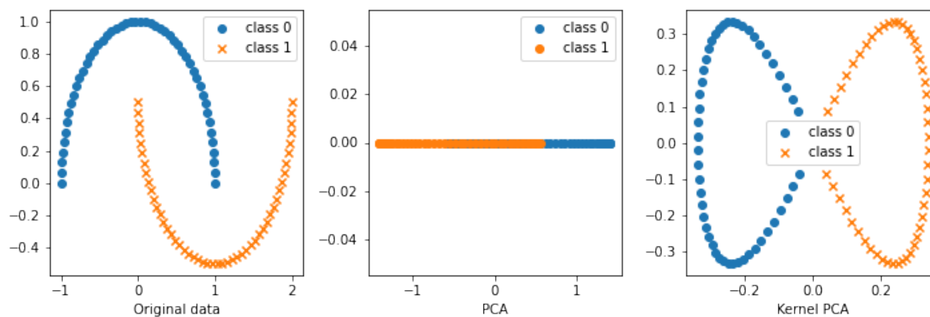


Figure 2.3: PCA and kPCA. The first graph shows the original data. The second graph plots the data transformed by PCA into 1D, which is obviously not linearly separable. The third plots the data transformed by kernel PCA into 2D space, which can be perfectly separated by a straight line. For the third graph, it also suffices to project data into 1D space as we can observe from x-axis.

Kernels are used in GPs as well and play a crucial role when we analyze the mean function and covariance function of a posterior distribution. We will talk more about

the neural networks with infinitely many neurons or channels as GPs in Section 2.4.1.

2.2.2 Eigenfunction Analysis of Kernels

For the purpose of analyzing the eigenfunctions of kernels, it is necessary to know the form of the integral operator T_k related to kernel $k(\cdot,\cdot)$:

$$[T_k f(\boldsymbol{x})] = \int k(\boldsymbol{x},\boldsymbol{x}')f(\boldsymbol{x}')d\mu(\boldsymbol{x}'), \tag{2.25}$$

where μ is a measure. Given a kernel function $k(\boldsymbol{x},\boldsymbol{x}'), \boldsymbol{x},\boldsymbol{x}' \in \mathcal{X} \subset \mathbb{R}^n$ and a function $\phi(\cdot)$, if we have

$$\int k(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x})d\mu(\boldsymbol{x}) = \lambda\phi(\boldsymbol{x}'), \tag{2.26}$$

then we call $\phi(\cdot)$ an eigenfunction of kernel $k(\cdot,\cdot)$ with eigenvalue λ with respect to measure μ. Note that, depending on the function space, there can be infinitely many eigenfunctions and any two different eigenfunctions are orthogonal with respect to the measure μ, namely

$$\int \phi_i(\boldsymbol{x})\phi_j(\boldsymbol{x})d\mu(\boldsymbol{x}) = \langle\phi_i,\phi_j\rangle_\mu = \begin{cases} 1, & \text{if } i=j; \\ 0, & \text{otherwise.} \end{cases} \tag{2.27}$$

With the above definitions, a connection among kernels, eigenfunctions and eigenvalues can be described as follows:

Theorem 2.1 (Mercer's Theorem) *Let $(\mathcal{X},\mu)$ be a finite measure space and $k \in L_\infty(\mathcal{X}^2,\mu^2)$ be a kernel function such that integral operator $T_k : L_2(\mathcal{X},\mu) \to L_2(\mathcal{X},\mu)$ defined by Equation 2.25 is positive definite, i.e.*

$$\int k(\boldsymbol{x},\boldsymbol{x}')f(\boldsymbol{x})f(\boldsymbol{x}')d\mu(\boldsymbol{x})d\mu(\boldsymbol{x}') > 0 \tag{2.28}$$

holds for any $f(\cdot) \in L_2(\mathcal{X},\mu)$. Assume $k(\cdot,\cdot)$ is a symmetric positive semidefinite kernel, then there exists orthogonal eigenfunctions of T_k associated with the eigenvalue $\lambda_i > 0$ such that

$$k(\boldsymbol{x},\boldsymbol{x}') = \sum_{i=1}^{+\infty} \lambda_i\phi_i(\boldsymbol{x})\phi_i(\boldsymbol{x}') \tag{2.29}$$

converges absolutely and uniformly.

If a kernel has only finite number of nonzero eigenvalues, then it is degenerate. If the measure μ poses weights on finite number of n data points, then the eigendecomposition of kernels will have only n eigenfunctions even if they are nondegenerate.

Interestingly, for heat conduction and diffusion the form of heat kernel is different from what implies by Mercer's theorem on account of the term $e^{-\lambda_n t}$ not being eigenvalue of $\frac{\partial}{\partial t} - \mathcal{L}$ but that of $e^{(\frac{\partial}{\partial t} - \mathcal{L})}$, and heat kernel is represented by the following:

$$K(t,x,y) = \sum_{n=0}^{\infty} e^{-\lambda_n t}\phi_n(x)\phi_n(y), \tag{2.30}$$

where $e^{-\lambda_n t}$ and $\phi_n(\cdot)$ are eigenvalue and eigenfunction, respectively, and $K(t,x,y)$ is the solution of

$$\frac{\partial K(t,x,y)}{\partial t} = \Delta_x K(t,x,y), \tag{2.31}$$

where Δ is the Laplacian operator. The reason for this representation is arose from that the associated integral transform has a form of $T = e^{t\Delta}$. Aside from the heat kernels, Jacot et. al. [16] also argued that the evolution of neural networks during training could be described by neural tangent kernel (NTK) parametrized by $\boldsymbol{\theta} \in \mathbb{R}^P$:

$$\Theta(x,y;\boldsymbol{\theta}) = \left\langle \partial_{\theta_p} f(x;\boldsymbol{\theta}), \partial_{\theta_p} f(y;\boldsymbol{\theta}) \right\rangle = \sum_{p=1}^{P} \partial_{\theta_p} f(x;\boldsymbol{\theta}) \partial_{\theta_p} f(y;\boldsymbol{\theta}), \tag{2.32}$$

in which $f(\cdot;\boldsymbol{\theta})$ is the scalar output of networks, $\partial_{\theta_p} f(\cdot)$ is the partial derivative with respect to θ_p and $\boldsymbol{\theta}$ is the vector of parameters of neural networks. Equation 2.32 can be extended to a multi-output situation as well. NTK converges to an explicit limiting kernel and stays constant during training in the limit of infinite network width. Both the neural tangent kernels and heat kernel show the evolution of certain processes: neural tangent kernel indicates evolution of networks while heat kernels represents the evolution of temperature in a certain region.

Together with integral operator T_k and Mercer's theorem, we can analyze eigenfunctions and eigenvalues of the operator. Now that we know a kernel function can be decomposed to an infinite sum of products of eigenfunctions and eigenvalues, we need methods to find what mathematical forms eigenfunctions and eigenvalues take, respectively. Using one dimensional RBF $k(x,x') = \exp\left(-\frac{(x-x')^2}{2l^2}\right)$ as an instance and presuming $x \sim \mathcal{N}\left(0,\sigma^2\right)$, we have analytical solutions for eigenvalues [25]:

$$\lambda_i = \sqrt{\frac{2a}{A}} B^i, \tag{2.33}$$

and eigenfunctions:

$$\phi_i(x) = \exp\left(-(c-a)x^2\right) H_i(\sqrt{2c}x), \tag{2.34}$$

where $H_i = (-1)^i \exp\left(x^2\right) \frac{d^i}{dx^i} \exp(-x^2)$, $a^{-1} = 4\sigma^2$, $b^{-1} = 2l^2$, $c = \sqrt{a^2 + 2ab}$, $A = a + b + c$, and $B = \frac{b}{A}$. If analytical results are not available, we can use numeric approximations. Given probabilistic measure $d\mu(\boldsymbol{x}) = p(\boldsymbol{x})d\boldsymbol{x}$, we have approximation

$$\begin{aligned} \lambda_i \phi_i(\boldsymbol{x}') &= \int k(\boldsymbol{x},\boldsymbol{x}') p(\boldsymbol{x}) \phi_i(\boldsymbol{x}) d\boldsymbol{x} \\ &\approx \frac{1}{n} \sum_{l=1}^{n} k(\boldsymbol{x}_l, \boldsymbol{x}') \phi_i(\boldsymbol{x}_l), \end{aligned} \tag{2.35}$$

where $\boldsymbol{x}_l$'s are samples from $p(x)$. Insert $\boldsymbol{x}' = \boldsymbol{x}_l, l = 1, \ldots, n$ into Equation 2.35, we have

$$\mathbf{K}\boldsymbol{u}_i = \lambda_i^{mat} \boldsymbol{u}_i, \tag{2.36}$$

where $\mathbf{K}$ is the kernel matrix, λ_i^{mat} is the ith matrix eigenvalue and $\boldsymbol{u}_i$ is the corresponding eigenvector. Consequently, we have

$$\phi_i(\boldsymbol{x}_j) \approx \sqrt{n}(\boldsymbol{u}_i)_j, \tag{2.37}$$

and

$$\lambda_i \approx \frac{\lambda_i^{mat}}{n}, \tag{2.38}$$

for $i = 1, \ldots, n$. We could also obtain approximated eigenfunctions by Nyström method as

$$\phi_i(\boldsymbol{x}') \approx \frac{\sqrt{n}\boldsymbol{k}(\boldsymbol{x}')^T \boldsymbol{u}_i}{\lambda_i^{mat}}, \tag{2.39}$$

where $\boldsymbol{k}(\boldsymbol{x}') = (k(\boldsymbol{x}_1, \boldsymbol{x}'), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}'))$. From [33], the eigendecomposition of kernel matrix could be done on a smaller system of size $m < n$, and then expanding the results back to n dimensions, which requires only $\mathcal{O}(m^2 n)$ operations. In [8], authors point out that m increases more slowly than data size n and the rate of convergence for variational Gaussian process regression depends on the decay of eigenvalues of the covariance operator.

In [7], the author has studied convergence properties of eigenfunctions and eigenvalues of kernel matrices and derived accurate bounds on approximation error, which has the property that the error bounds scale with the magnitude of an eigenvalue, namely the approximation errors of small eigenvalues are much smaller than those of large eigenvalues.

In Section 3, we will focus on eigenfunctions and eigenvalues of the linear operator T_k on kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ with entries $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

2.3 Nonlinear Dimensionality Reduction

In this section, we will discuss some nonlinear dimensional reduction methods that have the capability of finding the low dimensional manifold of data in a high dimensional space by employing spectral properties. Spectral clustering is a big family that makes use of spectrum to perform dimensionality reduction prior to clustering in low dimensional space, for a detailed explanation [30] would be a good choice. Except for (k)PCA we talked about in Section 2.2.1, there are also various techniques, such as Laplacian eigenmaps, isometric mapping (Isomap), and diffusion maps, and we will briefly introduce each of them.

Laplacian eigenmaps [3] consider the intrinsic geometry of data and builds graphs based on neighborhood information. Every data point in the data set can be regarded as a node and the neighborhoods of each node must be found by certain algorithms (such as the most commonly used K-nearest neighbor algorithm) such that the edge is decided by the distance between two nodes. As such, the graph we build can be considered as an approximation of the low dimensional manifold in the high dimensional space. Through the optimization of a cost function that maintains the connected points on the graph as close as possible, the local distance between two data points is preserved in the space.

With the help of Laplace-Beltrami operator, we can apply spectral decomposition to the corresponding graph Laplacian and interpret eigenvectors as low dimensional embedding. Figure 2.4 exhibits the reduced 2D data of MNIST images. The MNIST data set is imported from the sklearn library and contains 1797 grayscale images with a size of 8×8 pixels. By the power of Laplacian eigenmaps we reduce the 64-dimensional data into 2-dimensional data and the locality of data is maintained.

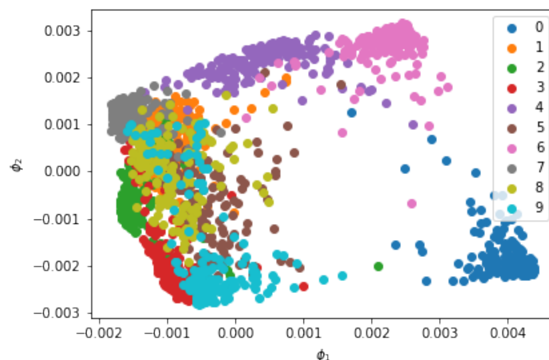


Figure 2.4: Laplacian eigenmaps applied on MNIST data. Varying clusters are denoted with differing colors, and each cluster represents one handwritten digit, which results in 10 clusters total in the graph.

Isomap [29] is another approach to seek a low dimensional manifold for data, and it extends the multidimensional scaling [5] method (MDS) by using geodesic distance induced by a neighborhood graph other than using Euclidean distance as the pairwise distance between data points. As we do in Laplacian eigenmaps, we need first to use KNN to construct the neighborhood graph and compute the shortest path between each pair of nodes in order to obtain the geodesic distance. Isomap defines geodesic distance as the sum of edge weights along the shortest path between two nodes in the graph. After that, we use MDS to compute the low dimensional embedding, from which the top n eigenvectors of the geodesic distance matrix represent the coordinates we desire in the n dimensional Euclidean space. Figure 2.5 shows the reduced MNIST handwritten digits in 3-dimensional space. From the plot, we can see that our data is reduced from 64-dimensional space into a 3-dimensional space and the nonlinear local structure is preserved.

Isomap is a direct version of Laplacian eigenmaps in that it needs us to construct the full geodesic distance matrix. A weakness of the Isomap algorithm is that the approximation of the geodesic distance is not robust to noise perturbation [11].

Diffusion maps [10] find the low dimensional embedding of data by computing the eigenvectors and eigenvalues of a diffusion operator on data and it can overcome the disadvantage of noise perturbation within data. Diffusion maps discover the underlying manifold that describes the geometry of datasets by integrating local features. To formulate the procedure of diffusion maps, we first need to construct the similarity matrix

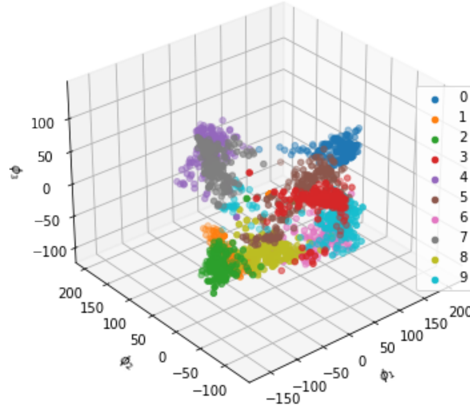


Figure 2.5: Isomap applied on MNIST data. Images are converted into 3D data and marked with same color if their labels are identical. Different clusters are distinguished by different colors in the legend shown in the right.

L, and then normalize the matrix to obtain matrix **P**. Next, we should diagonalize the matrix **P** and sort the eigenvectors and eigenvalues by descending order, ultimately the first n eigenvectors constitute the low dimensional embedding. Figure 2.6 shows the result after applying diffusion maps.

All the algorithms mentioned above employ spectral decomposition of a certain matrix that contains geometric information of data and utilize first n eigenvectors of such matrix as a low dimensional embedding, which is actually similar to what we will do in Section 3 except that we do eigendecomposition on the kernel matrix.

2.4 Gaussian Processes

Methods using a fixed number of parameters are parametric methods while methods using a non-fixed number of parameters but rather depending on the size of a dataset are called non-parametric methods. And Gaussian Processes (GPs) are non-parametric methods. To formulate Gaussian processes, we first state Gaussian distribution (also called normal distribution), since GPs are defined on the Gaussian distribution. Let x be a univariate Gaussian random variable, the probability density function (PDF) of a univariate normal distribution is

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{2.40}$$

where μ is the mean and σ^2 is the variance. The univariate Gaussian is widely used in statistical problems, such as regression and hypothesis test. For more general cases that

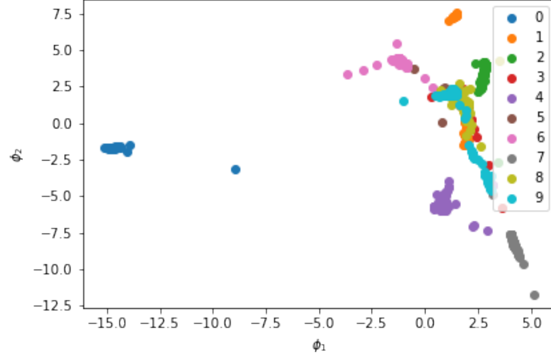


Figure 2.6: Diffusion map applied on MNIST data. Local geometry is ignored while the global geometry is preserved in the graph.

$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{pmatrix} \in \mathbb{R}^n$ follows independently identical distribution (i.i.d) for each entry x_i, we have PDF of the joint normal distribution:

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\boldsymbol{\Sigma}|} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right), \tag{2.41}$$

where the mean is

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{pmatrix}, \tag{2.42}$$

and the covariance matrix is

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_A & \boldsymbol{\Sigma}_{AB} \\ \boldsymbol{\Sigma}_{BA} & \boldsymbol{\Sigma}_B \end{pmatrix}. \tag{2.43}$$

If we condition $\boldsymbol{x}_A$ on $\boldsymbol{x}_B$, then we have conditional distribution

$$p(\boldsymbol{x}_A|\boldsymbol{x}_B) = \frac{p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\int p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\, d\boldsymbol{x}_A}, \tag{2.44}$$

with mean and variance

$$\mathbb{E}(\boldsymbol{x}_A|\boldsymbol{x}_B) = \boldsymbol{\mu}_A + \boldsymbol{\Sigma}_{AB} \boldsymbol{\Sigma}_B^{-1} (\boldsymbol{x}_B - \boldsymbol{\mu}_B), \tag{2.45}$$

$$\mathrm{Cov}(\boldsymbol{x}_A|\boldsymbol{x}_B) = \boldsymbol{\Sigma}_A - \boldsymbol{\Sigma}_{AB} \boldsymbol{\Sigma}_B^{-1} \boldsymbol{\Sigma}_{BA}, \tag{2.46}$$

which are quite helpful when we derive the posterior predictive distribution of GPs.

A Gaussian process defines a prior over functions, which can be converted into a posterior over functions when we have some data points. A GP is a collection of random variables such that joint distribution of every finite subsets of random variables $x_1, x_2, \ldots, x_N \in \mathbb{R}$ is a multivariate Gaussian. That is to say, given a random process $\{x_t\}_{t \in T}$, for any $\{t_1, \ldots, t_n\} \in T, n \geq 1$, $(x_{t_1}, \ldots, x_{t_n})$ is Gaussian random vector, then $\{x_t\}_{t \in T}$ is a GP with mean function

$$\mu(x_t) = \mathbb{E}(x_t), \tag{2.47}$$

and variance function

$$\Sigma(x_{t_1}, x_{t_2}) = k(x_{t_1}, x_{t_2}). \tag{2.48}$$

We can manually set the mean $\mu(\cdot) = 0$, since GP can perfectly fit the mean function, which means a GP is determined only by the covariance function $k(\cdot, \cdot)$. Like other methods, GPs can be used both for regression and classification. When solving regression problems, we have closed form solutions which can be computed in $\mathcal{O}(N^3)$, however, in classification problems we can only use approximation methods due to the non-Gaussianity of the posterior distribution. There are various advantages of Gaussian Processes, which include:

- The prediction interpolates the observations;
- Empirical confidence intervals can be computed to measure uncertainty;
- Various kernels can be used regarding concrete problems, and it is possible to specify custom kernels;
- It is easy to analyze;

while disadvantages are also obvious:

- It is very slow since GP uses whole samples to perform prediction and needs to compute matrix inverse;
- It is not efficient in high dimensional spaces;
- Constructing a new kernel might be not easy;
- GP largely ignores defining well-targeted objective functions (to minimize or maximize objective functions) and instead focuses on optimizing the marginal likelihood [25].

2.4.1 Covariance Functions

In Statistics, covariance is used to measure error of two random variables:

$$\begin{aligned} \mathrm{Cov}(x, y) &= \mathbb{E}\left[(x - \mathbb{E}(x))(y - \mathbb{E}(y))\right] \\ &= \mathbb{E}(xy) - \mathbb{E}(x)\mathbb{E}(y). \end{aligned} \tag{2.49}$$

Intuitively, if two random variables have the same trend of changing, the covariance is positive; if their trends are opposite to each other, the value will be negative. Except the covariance for random variables, it can be defined for a matrix as well. Let $\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n) \in \mathbb{R}^{m \times n}$ be a matrix, where $\boldsymbol{x}_i$ for $i = 1, \ldots, n$ is a column vector. Then we have the mean vector:

$$\boldsymbol{\mu}_X = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i}{n} = \frac{1}{n}\mathbf{X}\mathbf{1}_n, \tag{2.50}$$

where $\mathbf{1}_n = (1, \ldots, 1)^T \in \mathbb{R}^n$. And the covariance matrix is denoted as:

$$\mathbf{C}_X = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu}_X)(\boldsymbol{x}_i - \boldsymbol{\mu}_X)^T = \frac{1}{n} \mathbf{X} \mathbf{J}_n \mathbf{X}^T, \tag{2.51}$$

where $\mathbf{J}_n = \mathbf{I}_n - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n^T$ is the centering matrix. With the above covariance matrix, we can regard an image as a flattened matrix, where m is the number of channels and n is the number of pixels, and an image can be represented by a covariance matrix, which gives us the impression that representing an image by a covariance matrix is essentially equivalent to representing an image by a Gaussian probability density p in $\mathbb{R}^m$ with zero mean [19]. Feature vector $\boldsymbol{x}_i$ is a random observation of a m-dimensional random vector with probability density p. Figure 2.7 shows 10 different digits with each pixel being normalized in the range $[0, 1]$ and their corresponding covariance in the above. Each covariance matrix has only one entry due to one channel of the grayscale images. We see that any two different digits have disparities between their corresponding covariance. It would be very evident for us that the similar digits would have a similar covariance

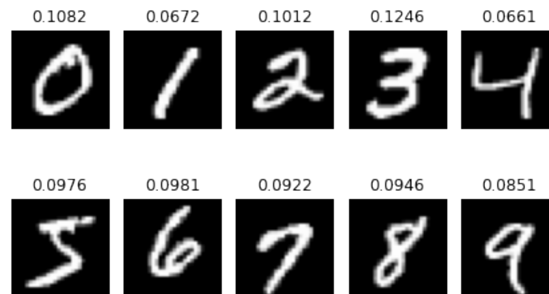


Figure 2.7: Handwritten digits and their corresponding covariance.

matrix as shown in Figure 2.8, from which we see that the covariance of the 3rd and 4th image in the first row and the 4th in the second row have very close covariance values due to same orientation, analogous shapes, and thickness of handwriting. Same rules apply for the first image in the first row and the first image in the second row. Obviously, covariance captures some important information from each image.

As for covariance of GPs, we know from previous sections that defining the covariance function $k(\cdot, \cdot)$ is defining the GP. A kernel function that satisfies the symmetry property is a covariance function. For simplicity, the terminologies kernel function and covariance function will be the same for subsequent contents from now on and symbol $k(\cdot, \cdot)$ will be used to refer to covariance functions. All the kernels we talked about in Section 2.2.1 can be used as covariance functions. Depending on the problem we face, we can construct new kernels from existing kernels by related rules mentioned before.

Generally speaking, a GP is stationary and non-stationary otherwise if the following conditions are satisfied:

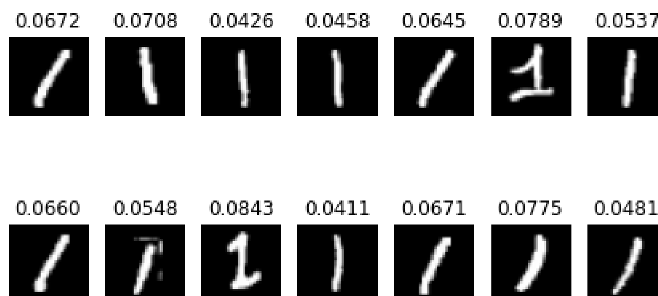


Figure 2.8: Digit 1's and corresponding covariance.

- $\mu(x_{t_1}) = \mu(x_{t_1+s})$ for $t_1, s \in T$;
- $k(x_{t_1}, x_{t_2}) = k(x_{t_1+s}, x_{t_2+s})$ for $t_1, t_2,$ and $s \in T$,

where $\mu(\cdot)$ is the mean function, $k(\cdot,\cdot)$ is the covariance function, and T is an index set. This implies that stationary kernel function takes the form $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} - \boldsymbol{x}')$. As we mentioned, the mean function is usually set to be zero due to the advantage of flexibility of GP, so we only need to focus on the covariance of GPs.

Next, we will discuss applications of GP for regression and classification, respectively.

2.4.2 Gaussian Processes for Regression

Assume the prior on regression function is a GP:

$$f(\boldsymbol{x}) \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')), \tag{2.52}$$

where $m(x)$ is the mean function and $k(x, x')$ is the kernel function, thus we have

$$m(\boldsymbol{x}) = \mathbb{E}\left[f(\boldsymbol{x})\right], \tag{2.53}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}\left[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))\right], \tag{2.54}$$

where $k(\cdot,\cdot)$ is positive definite. We then have a joint Gaussian distribution for a finite set of points $\mathcal{D} = \{(\boldsymbol{x_i}, y_i)\}$ for $i = 1, \ldots, N$:

$$p(\boldsymbol{f}|\mathbf{X}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{\mu}, \mathbf{K}), \tag{2.55}$$

where $\boldsymbol{\mu} = (\mu(\boldsymbol{x_1}), \ldots, \mu(\boldsymbol{x_N}))^T$ and $\mathbf{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. For GP regression, we have different representations for noise-free data and noise data, respectively.

GP regression for noise-free data

With noise-free training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, N\}$ and testing data $\mathcal{D}_* = \{(\boldsymbol{x}'_i, y'_i), i = 1, \ldots, N_*\}$, we have $f_i = f(\boldsymbol{x}_i)$ at data point $\boldsymbol{x}_i$ and $\boldsymbol{f}_*$ is the output of our test data. The joint distribution has the form:

$$\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right), \tag{2.56}$$

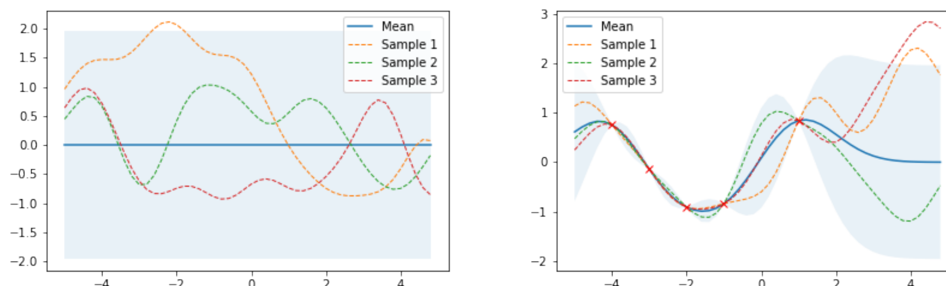
where $\mathbf{K} \in \mathbb{R}^{N \times N}$ with entry $\mathbf{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\mathbf{K}_* \in \mathbb{R}^{N \times N_*}$ and $\mathbf{K}_{**} \in \mathbb{R}^{N_* \times N_*}$. After conditioning, we have posterior:

$$p(\boldsymbol{f}_*|\mathbf{X}_*, \mathbf{X}, \boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \tag{2.57}$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1}(\boldsymbol{f} - \boldsymbol{\mu}(\mathbf{X})), \tag{2.58}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*. \tag{2.59}$$

In Figure 2.9, we compare samples and the 2 times standard deviation (marked with the shaded region) from GP prior and posterior. We can see from Figure 2.9a that when we sample from prior $p(\boldsymbol{f}|\mathbf{X})$ the standard deviation makes no difference for all 3 samples due to the same prior at each point, but when we condition on 5 points, the standard deviation of posterior $p(\boldsymbol{f}_*|\mathbf{X}_*, \mathbf{X}, \boldsymbol{f})$ at each point is really small comparing to the region without training points as shown in Figure 2.9b, which means we are more certain around training points when we take samples.



(a) Samples from prior distribution with mean 0 and standard deviation 1.

(b) Samples from posterior distribution with training points marked by red crosses.

Figure 2.9: Samples from prior and posterior of a Gaussian process with RBF kernel marked with 2 times standard deviation above and below.

GP regression for noise data

Consider the noisy data with function $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. We cannot interpolate the data perfectly due to the noise. We have the covariance of training data as:

$$\text{cov}(\boldsymbol{y}|\mathbf{X}) = \mathbf{K}_y = \mathbf{K} + \sigma_y^2 \mathbf{I}_N, \tag{2.60}$$

where $\mathbf{K}$ is the covariance matrix of $f(\boldsymbol{x})$ and $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ the identity. Assuming the mean are zeros, we thus have the joint distribution of training data and testing data:

$$\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_y & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right). \tag{2.61}$$

Consequently, the posterior distribution will be:

$$p\left(\boldsymbol{f}_* | \mathbf{X}_*, \mathbf{X}, \boldsymbol{y}\right) = \mathcal{N}(\boldsymbol{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \tag{2.62}$$

$$\boldsymbol{\mu}_* = \mathbf{K}_*^T \mathbf{K}_y^{-1} \boldsymbol{y}, \tag{2.63}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_*. \tag{2.64}$$

If we only have one testing point, the above results can be simplified to

$$p\left(f_* | \boldsymbol{x}_*, \mathbf{X}, \boldsymbol{y}\right) = \mathcal{N}(f_* | \mathbf{k}_*^T \mathbf{K}_y^{-1} \boldsymbol{y}, k_{**} - \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{k}_*), \tag{2.65}$$

where $\mathbf{k}_* = (k(\boldsymbol{x}_*, \boldsymbol{x}_1), \ldots, k(\boldsymbol{x}_*, \boldsymbol{x}_N))$ and $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$. From Figure 2.10 we see that the samples are not as smooth as that of posterior of noise-free data in Figure 2.9b, which is resulted from the noise added for the data.

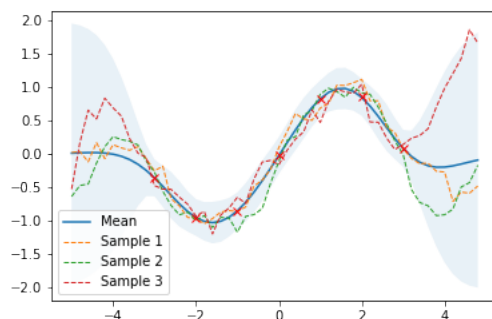


Figure 2.10: Posterior distribution of noisy data. Red crosses denote the training points. Uncertainty goes up at the region without training points. 3 samples drawn from posterior are not as smooth as those of Figure 2.9b due to noise.

Although we use the same kernel function for posterior distribution, the different values of kernel parameters influence the distribution as well. Due to the noisy data, the kernel has a slightly different representation from Equation 2.16, for which we add term $\sigma_y^2 \delta_{ij}$, where σ_y^2 is the variance of noise data, and $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. Figure 2.11 illustrates the impact of different parameters of RBF kernel on the samples and uncertainty. We can see from the first row that bigger length scale l gives us a smoother mean and samples and the uncertainty region of larger length scale l is not as bumped as the smaller one; the second row demonstrates that greater vertical variation coefficient σ_f will produce bigger standard deviation for the region without

training data and thus more uncertain, and the final row tells us that when noisy data have a higher variance σ_y^2 the values of samples fluctuate more fierce and we are more uncertain than the posterior that has a lower variance for noise. At the same time, the mean function cannot perfectly interpolate training points. Kernel parameters can be estimated by maximizing the marginal log likelihood, which is based on the gradient method, or we can use Bayesian inference to compute the posterior such that Monte Carlo is applied to approximate parameters.

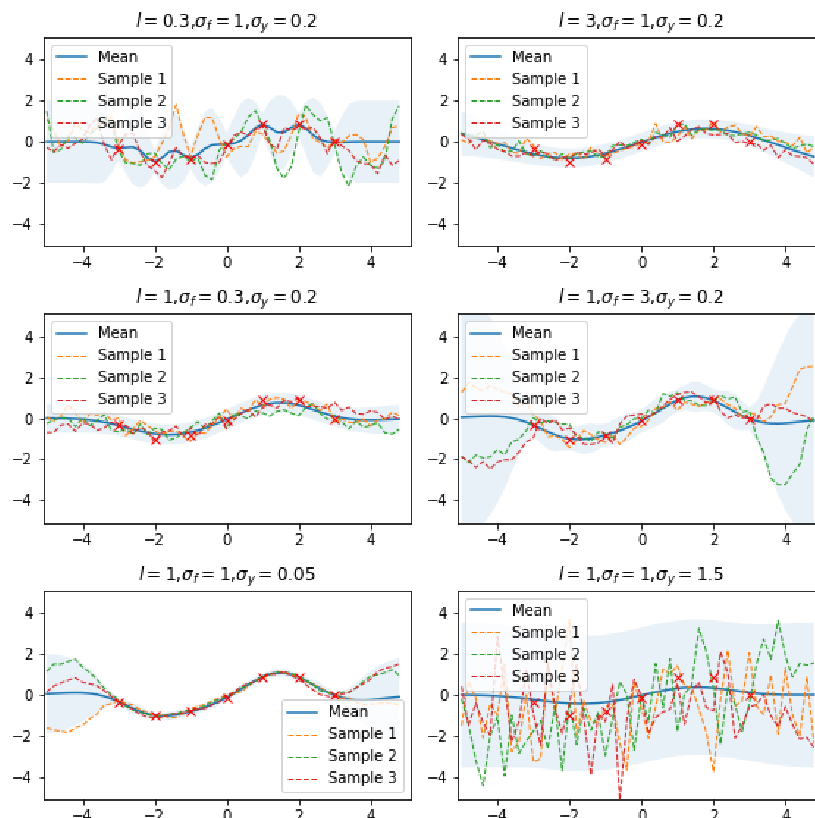


Figure 2.11: Influence of different values of parameters on RBF kernel. l is the length scale of RBF and σ_f is the vertical variation coefficient, and σ_y the standard deviation of noise data. The shaded region is the 2 times standard deviation above and below the mean. The wider the shaded region is, the more uncertain when we take samples. Red crosses are training points.

2.4.3 Gaussian Processes for Classification

Other than applying the Gaussian processes to regression, GP can also be used in classification for new data points. Here we discuss the binary classification and GP for multi-class classification. In Section 3, we will focus on GP for multi-class classification.

GP classification for binary class

The commonly used approach to classify data is to output a categorical distribution for classes. When it comes to binary classification, we only need to consider Bernoulli distribution, since the probability of points belonging to one class will be identical to the probability of not belonging to the other class, which indicates that it is sufficient for us to consider only the situation of points belonging to class +1.

Assume the binary labels are $y_i \in \{-1, +1\}$ and define a GP on $f(\boldsymbol{x}_i)$ for $\mathcal{D} = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, N\}$, the probability of point $\boldsymbol{x}_i$ belonging to class +1 is

$$p(y_i = +1|\boldsymbol{x}_i) = \sigma(y_i f(\boldsymbol{x}_i)), \tag{2.66}$$

and the probability of point $\boldsymbol{x}_i$ belonging to class -1 will be

$$p(y_i = -1|\boldsymbol{x}_i) = 1 - p(y_i = +1|\boldsymbol{x}_i), \tag{2.67}$$

where $\sigma(\cdot)$ is the sigmoid function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{2.68}$$

Thus we obtain a non-Gaussian stochastic process over function $p \in (0, 1)$.

Given the dataset $\mathcal{D}$, we need to determine the predictive distribution $p(y_{N+1}|\boldsymbol{x}_{N+1}, \mathbf{X}_N, \boldsymbol{y}_N)$ for unseen point $\boldsymbol{x}_{N+1}$, where $\mathbf{X}_N = (\boldsymbol{x}_i, \ldots, \boldsymbol{x}_N)$ and $\boldsymbol{y}_N = (y_1, \ldots, y_N)$. The Gaussian process prior for $\boldsymbol{f}_N = (f_1, \ldots, f_N)$ takes the form:

$$p(\boldsymbol{f}_{N+1}) = \mathcal{N}(\boldsymbol{f}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}), \tag{2.69}$$

where the covariance matrix $\mathbf{C}_{N+1} \in \mathbb{R}^{(N+1)\times(N+1)}$ does not include noise term since we assume each data point is labeled correctly. Nevertheless, for numeric stability it is better to introduce some noise into our data which ensures that the covariance matrix is PSD, and the entries of $\mathbf{C}_{N+1}$ is given by

$$\mathbf{C}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \nu\delta_{ij}, \tag{2.70}$$

where $k(\cdot, \cdot)$ is a PSD kernel function and ν is usually a fixed value.

As we mentioned, it is adequate to predict $p(y_{N+1} = +1|\boldsymbol{y}_N)$ and the related predictive distribution is

$$p(y_{N+1} = +1|\boldsymbol{y}_N) = \int p(y_{N+1} = +1|f_{N+1})p(f_{N+1}|\boldsymbol{y}_N)df_{N+1}, \tag{2.71}$$

where $p(y_{N+1} = +1|f_{N+1}) = \sigma(f_{N+1})$. Equation 2.71 is not analytically tractable, but approximation methods could be applied to obtain solutions. For example, we could use

$$\int \sigma(a)\mathcal{N}(a|\mu, \sigma^2)da \approx \sigma(k(\sigma^2)\mu), \tag{2.72}$$

where $k(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2}$, to approximate Equation 2.71. However, Equation 2.72 requires the approximation of posterior distribution $p(f_{N+1}|\boldsymbol{y}_N)$ to be Gaussian. Laplacian approximation could be used to seek a Gaussian approximation of the posterior distribution over f_{N+1}, which has

$$\mathbb{E}\left[f_{N+1}|\boldsymbol{y}_N\right] = \boldsymbol{k}^T(\boldsymbol{y}_N - \boldsymbol{\sigma}_N), \tag{2.73}$$

$$\mathrm{Cov}\left[f_{N+1}|\boldsymbol{y}_N\right] = c - \boldsymbol{k}^T(\mathbf{W}_N^{-1} + \mathbf{C}_N)^{-1}\boldsymbol{k}, \tag{2.74}$$

where $\boldsymbol{k} = (k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1}))^T$, $\boldsymbol{\sigma}_N = (\sigma(f_1), \ldots, \sigma(f_N))^T$ and $\mathbf{W}_N = \mathrm{diag}(\sigma(f_1)(1-\sigma(f_1)), \ldots, \sigma(f_N)(1-\sigma(f_N)))$. The parameters of the covariance function still need to be determined, for which we could maximize the likelihood function in order to obtain desired results. Other methods, such as variational inference and expectation propagation, have good results for classification as well.

In order to illustrate the Gaussian processes for binary classification, we use the data generated by function **make_moons**() from Python library sklearn. We set noise to 0.3 and generate 500 points with labels $\{0, 1\}$ marked with corresponding legends as shown in Figure 2.12. We choose 0.5 as the boundary value of classification and use also RBF kernel with length scale $l = 1$ and vertical variation coefficient $\sigma_f = 1$, hence we get the predicted classification result in the figure. Although there are some incorrectly classified points that result from noise parameters, generally speaking, we get good predictions for class labels. Note that when the noise is higher, there will be more points classified incorrectly.

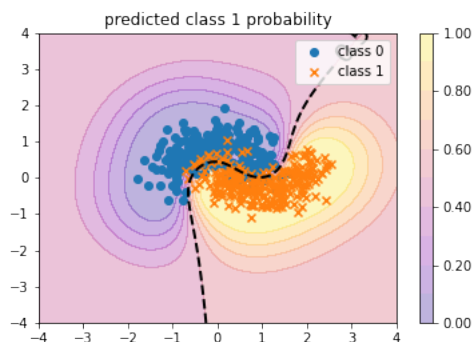


Figure 2.12: Gaussian processes for binary classification on 500 data points. Black dashed line is the boundary of classification, for which we use 0.5. Colored contours mark the probabilities of points being classified to class 1, as shown in the color bar on the right.

GP classification for multi-class

The Laplacian approximation for binary classification can be extended to $K > 2$ classes very intuitively, if we use the softmax function rather than the sigmoid function in Equation 2.68. The softmax function (also known as normalized exponential function)

is a generalization of logistic function to multiple dimensions and represented by

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_k)}, \tag{2.75}$$

where K is the number of classes. Alternatively, we can use the GP classification methods for binary classification multiple times to realize multiclass classification. There are many libraries provide sophisticated methods to do multiclass classification by one-vs-rest such as sklearn [24] and pyGPs [22]. Figure 2.13 shows the classification results for iris dataset using one-vs-rest method and good classification results for the data are obeserved.

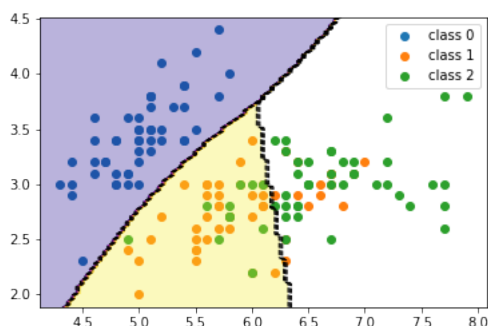


Figure 2.13: Gaussian processes for multiclass classification through one-vs-rest in pyGPs applied on 2D iris dataset. Black dashed lines are the boundaries that separate points belonging to different classes, and each class region is marked with a distinct color.

In [13], the authors regarded the classification problem as a regression problem without noise and encoded the labels using one-hot vectors in $\mathbb{R}^N$ with entries $\{-1, +1\}$, for which the label vector will have $+1$ only at the position where it belongs to the class and the rest are set to -1. As a result, the predicted class will take the index which corresponds to the highest value of the computed mean vector. We will use the same methods here to predict class labels for MNIST images in Section 3.

2.5 Neural Networks

Neural networks are comparatively important models in deep learning, and they have been successfully instantiated that they have immense potential in decision making, image analysis, pattern recognition, and data mining, etc.

Fully connected networks (FCN) is one of the most frequently used models for regression and classification. Hornik [15] has studied the properties of FCN with one hidden layer and proposed:

Theorem 2.2 (Universal Approximation Theorem) *For a continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ and positive integers d, D, the function σ is not a polynomial if and only if for*

every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$*, every compact subset* K *of* $\mathbb{R}^d$ *and every* $\epsilon > 0$ *there exists a continuous function* $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ *with representation*

$$f_\epsilon = W_2 \circ \sigma \circ W_1, \tag{2.76}$$

where W_2, W_1 *are composable affine maps and* $\circ$ *denotes component-wise composition such that the approximation bound*

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \epsilon \tag{2.77}$$

holds for any ϵ *arbitrarily small.*

For more details, please refer to [15]. According to Theorem 2.2, when we choose a fixed ϵ, we can find as many neurons as needed in a network to fit a continuous function. That is to say, a neural network with linear output and one hidden layer can approximate any continuous function defined over a closed and bounded subset of $\mathbb{R}^d$, if the above conditions are satisfied. The good news is there always exists a single hidden layered network that can approximate any function for a precision ϵ, but downsides also exist since the learning algorithm is not guaranteed to find the optimal parameters, which may cause overfitting or result in high training loss. Plus, single hidden layered networks will need a huge amount of neurons to reach a fixed precision while multilayered perceptron (MLP, also called FCN) just need to add more layers and the number of total neurons is far smaller compared to a single-layered network. Nonetheless, not the more layers, the better it is. In the Ph.D. thesis of Duvenaud [12], it has been shown that as the number of layers increases, the amount of information retained about the original input diminished to a single degree of freedom, since the largest singular values of the Jacobian of a set of functions drawn from independent GP priors tends to dominate as networks go deeper. This implied that with high probability, these functions vary little in all directions but one, making them unsuitable for computing representations of manifolds of more than one dimension. An interesting fact to be noticed is that we always first determine the architecture of a network and then compute the accuracy and loss of the network, which is actually opposite to the procedure stated by Theorem 2.2.

Moreover, the relationship between accuracy and training size is kind of interesting, since in the limit of training size going to infinity the test data size becomes zero, which means all the data are fed to networks and our networks have adequate ability to identify every possible occurrence, causing a 100% accuracy. Likewise, we could let neurons of networks be infinite, and discuss the limit of such situation next.

2.5.1 Neural Networks as Gaussian Processes

There have been numerous researches that investigate the relationships between neural networks with infinite neurons or channels and Gaussian processes. We will start from the simpler network architecture to more complicated ones.

Bayesian Neural Networks

From Neal's thesis [21], we know that with a wide class of weight priors over functions a Bayesian neural network in the limit of an infinite number of hidden neurons tends to be a GP.

Given a network with one hidden layer of N neurons, the network takes input $\boldsymbol{x} \in \mathbb{R}^n$ and outputs a single value $f(\boldsymbol{x}) \in \mathbb{R}$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_{2i} h(\boldsymbol{w}_{1i}; \boldsymbol{x}) + b, \tag{2.78}$$

where w_{2i} is the hidden-to-output weight of neuron i, $\boldsymbol{w}_{1i}$ is the weight vector of neuron i of input layer, and $h(\cdot)$ is the activation function applied for each neuron depending on $\boldsymbol{w}_{1i}$ and b the bias. Let b and w_{2i} have independent zero mean distributions of variance of σ_b^2 and $\sigma_{w_2}^2$, respectively, and let weight vector $\boldsymbol{w}_{1i}$ for each neuron be i.i.d. Denoting all weights by $\mathbf{W}$, we obtain [21]

$$\mathbb{E}_W(f(\boldsymbol{x})) = 0, \tag{2.79}$$

$$\begin{aligned} \mathbb{E}_W\left[f(\boldsymbol{x})f(\boldsymbol{x}')\right] &= \sigma_b^2 + \sum_{i=1}^{N} \sigma_{w_2}^2 \mathbb{E}_{w_{1i}}\left[h(\boldsymbol{w}_{1i}; \boldsymbol{x})h(\boldsymbol{w}_{1i}; \boldsymbol{x}')\right] \\ &= \sigma_b^2 + N\sigma_{w_2}^2 \mathbb{E}_{w_{1i}}\left[h(\boldsymbol{w}_{1i}; \boldsymbol{x})h(\boldsymbol{w}_{1i}; \boldsymbol{x}')\right]. \end{aligned} \tag{2.80}$$

The final term of Equation 2.80 becomes $\omega^2 \mathbb{E}_{w_{1i}}\left[h(\boldsymbol{w}_{1i}; \boldsymbol{x})h(\boldsymbol{w}_{1i}; \boldsymbol{x}')\right]$ if we scale $\sigma_{w_2}^2$ as ω^2/N. As the activation function $h(\cdot)$ is bounded, all moments of the distribution are bounded and hence the central limit theorem could be applied, showing that the stochastic process will converge to a GP in the limit as $N \to \infty$.

Deep Neural Networks

Not only converges one-layered neural networks to GPs, but also multiple layer neural networks with infinite neurons for each layer have a limit of GPs. In the work of Lee et. al. [18], they derived the exact equivalence between infinitely wide deep networks and Gaussian Processes. Consider a fully connected neural network of L layers with layer width N_l and pointwise nonlinear activation function $\phi(\cdot)$. Let $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ be the input vector and $\boldsymbol{z}^L \in \mathbb{R}^{d_{out}}$ be the output. The post and pre-activations of component i at layer l are denoted by x_i^l and z_i^l, respectively. Weights W_{ij}^l and biases b_i^l of layer l are drawn from independently identical distributions with zeros and variance σ_w^2/N_l and respectively, σ_b^2. We can formulate our network as:

$$x_j^l(\boldsymbol{x}) = \phi(z_j^{l-1}(\boldsymbol{x})), \tag{2.81}$$

$$z_i^l(\boldsymbol{x}) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l x_j^l(\boldsymbol{x}), \tag{2.82}$$

where $x_j^0(\boldsymbol{x}) = x_j$ and $N_0 = d_{in}$ for $l = 0$. We know from deduction of Bayesian neural networks that z_i^1 is a Gaussian process if $N_1 \to \infty$ with mean μ^1 in Equation 2.79 and

variance K^1 in Equation 2.80, denoted as $z_i^1 \sim GP(\mu^1, K^1)$. We should note that any two z_i^1, z_j^1 are joint Gaussian and have 0 covariance, thus ensuring the independence despite utilizing same features produced by the hidden layer. As $N_1 \to \infty, N_2 \to \infty, \ldots, N_L \to \infty$ in succession, it guarantees the input of each layer is governed by a GP.

Suppose z_j^{l-1} is a GP, which is i.i.d for every j and hence we have i.i.d $x_j^l(\boldsymbol{x})$'s. After $l-1$ steps, $z_i^l(\boldsymbol{x})$ is a sum of i.i.d random terms as $N_l \to \infty$ such that $z_i^l \sim GP(0, K^l)$ with covariance

$$
\begin{aligned}
K^l(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\left[z_i^l(\boldsymbol{x}), z_i^l(\boldsymbol{x}')\right] \\
&= \sigma_b^2 + \sigma_w^2 \mathbb{E}_{z_i^{l-1} \sim GP(0, K^{l-1})}\left[\phi(z_i^{l-1}(\boldsymbol{x}))\phi(z_i^{l-1}(\boldsymbol{x}'))\right] \\
&= \sigma_b^2 + \sigma_w^2 F_\phi\left(K^{l-1}(\boldsymbol{x}, \boldsymbol{x}), K^{l-1}(\boldsymbol{x}, \boldsymbol{x}'), K^{l-1}(\boldsymbol{x}', \boldsymbol{x}')\right),
\end{aligned}
\tag{2.83}
$$

where the joint distribution of $z_i^{l-1}(\boldsymbol{x})$ and $z_i^{l-1}(\boldsymbol{x}')$ has a covariance described by $K^{l-1}(\boldsymbol{x}, \boldsymbol{x})$, $K^{l-1}(\boldsymbol{x}, \boldsymbol{x}')$, and $K^{l-1}(\boldsymbol{x}', \boldsymbol{x}')$, and F_ϕ is a deterministic function that depends on ϕ. With the above iterative formula, we could obtain K^L for the GP describing the final output. Assume $W_{ij}^0 \sim \mathcal{N}(0, \sigma_w^2/d_{in})$ and $b_i^0 \sim \mathcal{N}(0, \sigma_b^2)$, we have base covariance

$$
\begin{aligned}
K^0(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}\left(z_i^0(\boldsymbol{x}) z_i^0(\boldsymbol{x}')\right) \\
&= \sigma_b^2 + \sigma_w^2\left(\frac{\boldsymbol{x}^T\boldsymbol{x}'}{d_{in}}\right),
\end{aligned}
\tag{2.84}
$$

which is a special case for Equation 2.80 if we consider $h(\boldsymbol{w}_{1i}; \boldsymbol{x})$ as a identity map $\mathcal{I}$ on $\boldsymbol{w}_{1i}^T\boldsymbol{x}$.

Convolutional Neural Networks

Convolutional neural networks are weight-sharing architectures that apply convolution filters to extract features from input data, especially for image data. Rasmussen et. al [13] shows that the output of a convolutional neural network with appropriate priors over weights and biases is a GP in the limit of infinite convolutional filters, which holds for residual convolutional neural networks as well. We will follow the procedure in the paper.

Let $\mathbf{X}$ be an input image of height $H^{(0)}$ and width $D^{(0)}$ with $C^{(0)}$ channels. We flatten the image data to a matrix of size $C^{(0)} \times (H^{(0)}D^{(0)})$ and denote each row as $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{C^{(0)}}$. Consider a network with L hidden layers, then the first layer activation $\mathcal{A}^{(1)}(\mathbf{X})$ is a linear map of input images, defined as

$$
\boldsymbol{\alpha}_i^{(1)}(\mathbf{X}) = b_i^{(1)}\mathbf{1} + \sum_{j=1}^{C^{(0)}} \mathbf{W}_{ij}^{(1)}\boldsymbol{x}_j,
\tag{2.85}
$$

where $\mathbf{1}$ is a all one vector in $\mathbb{R}$, for $i \in \{1, \ldots, C^{(1)}\}$, and other activations from $\mathcal{A}^{(2)}(\mathbf{X})$

to $\mathcal{A}^{(L+1)}(\mathbf{X})$ are defined recursively

$$\boldsymbol{\alpha}_i^{(l+1)}(\mathbf{X}) = b_i^{(l+1)}\mathbf{1} + \sum_{j=1}^{C^{(l)}} \mathbf{W}_{ij}^{(l+1)}\phi\left(\boldsymbol{\alpha}_j^{(l)}(\mathbf{X})\right), \tag{2.86}$$

where $\mathbf{W}_{ij}^{(l)}$ is the corresponding transformed matrix of ith filter $\mathbf{U}_{ij}^{(l)}$ at channel j on layer l and $\phi(\cdot)$ is applied elementwise. The activation $\mathcal{A}^{(l)}(\mathbf{X})$ are $C^{(l)} \times (H^{(l)}D^{(l)})$ matrices. For the regression and classification problems, we have $H^{(L+1)} = D^{(L+1)} = 1$, which is equivalent to a fully connected output layer. For each layer l, assume we have:

$$U_{i,j,x,y}^{(l)} \sim \mathcal{N}\left(0, \sigma_w^2/C^{(l)}\right), b_i^{(l)} \sim \mathcal{N}\left(0, \sigma_b^2\right), \tag{2.87}$$

where x, y is a location within the filter.

To justify that the output of the network above is a GP, as in deep neural networks the multivariate central limit theorem must be applied to each layer.

Consider a vector from two images $\mathbf{X}$ and $\mathbf{X}'$ of the form

$$\boldsymbol{\alpha}_i^{(l)}(\mathbf{X}, \mathbf{X}') = \begin{pmatrix} \boldsymbol{\alpha}_i^{(l)}(\mathbf{X}) \\ \boldsymbol{\alpha}_i^{(l)}(\mathbf{X}') \end{pmatrix}. \tag{2.88}$$

For any pair of data points $\mathbf{X}$ and $\mathbf{X}'$, Equation 2.88 is multivariate Gaussian jointly distributed for layer 1, due to the shared Gaussian biases and filters in Equation 2.87. Following Equation 2.85, we have

$$\boldsymbol{\alpha}_i^{(1)}(\mathbf{X}, \mathbf{X}') = b_i^{(1)}\mathbf{1} + \sum_{i=1}^{C^{(0)}} \begin{pmatrix} \mathbf{W}_{ij}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{ij}^{(1)} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_i \\ \boldsymbol{x}_i' \end{pmatrix}, \tag{2.89}$$

where $\boldsymbol{\alpha}_i^{(1)}(\mathbf{X}, \mathbf{X}')$ and $\boldsymbol{\alpha}_{i'}^{(1)}(\mathbf{X}, \mathbf{X}')$ are i.i.d for different filters. If we let the number of channels at layer l go to infinity, the feature maps $\boldsymbol{\alpha}_i^{(l+1)}(\mathbf{X}, \mathbf{X}')$ at layer $l+1$ will also be i.i.d multivariate Gaussian. By applying Equation 2.86, we have

$$\boldsymbol{\alpha}_i^{(l+1)}(\mathbf{X}, \mathbf{X}') = b_i^{(l+1)}\mathbf{1} + \sum_{j=1}^{C^{(l)}} \begin{pmatrix} \mathbf{W}_{ij}^{(l+1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{ij}^{(l+1)} \end{pmatrix} \phi\left(\boldsymbol{\alpha}_j^{(l)}(\mathbf{X}, \mathbf{X}')\right), \tag{2.90}$$

from which the first term is multivariate Gaussian as we assumed and the second term is infinite sum of i.i.d terms as $C^{(l)} \to \infty$ and also Gaussian. Therefore, output $\mathcal{A}^{(l+1)}(\mathbf{X}, \mathbf{X}')$ at layer $l+1$ is a joint multivariate Gaussian.

Other neural networks could be regarded as Gaussian processes as well. For instance, in [14], the authors showed that neural kernel tensor network and tensor network hidden layer neural network will converge to the Gaussian process as the width of each network goes to infinity. In [23], the authors derived the equivalence between Gaussian processes and multi-layer convolutional neural networks with and without pooling layers, and proved that the GPs corresponding to CNNs with and without weight sharing are identical in the absence of pooling layers.

2.5.2 Kernels of Neural Networks

We will discuss concrete forms of covariance function corresponding to the networks mentioned above.

Kernel of Bayesian Neural Network

Based on the Bayesian network structure represented by Equation 2.78, if we can evaluate the term $\mathbb{E}_{w_{1i}}\left[h(\boldsymbol{w}_{1i};\boldsymbol{x})h(\boldsymbol{w}_{1i};\boldsymbol{x}')\right]$ we then can have the covariance function of the network. The concrete form of covariance function depend on the choice of function $h(\cdot)$. Williams [32] has shown the closed form of kernel functions of Gaussian processes corresponding to networks with sigmoidal and gaussian hidden units. Using the error function erf(z) $= \frac{2}{\pi}\int_0^z e^{-t^2}dt$ as activation function, $h(\boldsymbol{w}_{1i};\boldsymbol{x}) = \mathrm{erf}(w_{1i,0} + \sum_{j=1}^{n} w_{1i,j}x_j)$, and setting $\boldsymbol{w}_{1i} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$, we get

$$k_{erf}(\boldsymbol{x},\boldsymbol{x}') = \frac{2}{\pi}\sin^{-1}\left(\frac{2\tilde{\boldsymbol{x}}^T\boldsymbol{\Sigma}\tilde{\boldsymbol{x}}'}{\sqrt{(1+2\tilde{\boldsymbol{x}}^T\boldsymbol{\Sigma}\tilde{\boldsymbol{x}})(1+2\tilde{\boldsymbol{x}}'^T\boldsymbol{\Sigma}\tilde{\boldsymbol{x}}')}}\right), \tag{2.91}$$

where $\tilde{\boldsymbol{x}} = (1, x_1, \ldots, x_n)$ is an augmented input vector. The Tanh function $k(\boldsymbol{x},\boldsymbol{x}') = \tanh(a + b\boldsymbol{x}^T\boldsymbol{x}')$ was also proposed, but it is not positive definite, thus it cannot be used as a valid kernel function [27]. Williams also set $h(\boldsymbol{w}_{1i};\boldsymbol{x}) = \exp(-\frac{(\boldsymbol{x}-\boldsymbol{w}_{1i})^T(\boldsymbol{x}-\boldsymbol{w}_{1i})}{2\sigma_g^2})$ with width factor σ_g^2 and $\boldsymbol{w}_{1i} \sim \mathcal{N}(\boldsymbol{0}, \sigma_{w_1}^2\boldsymbol{I})$, and obtained

$$\begin{aligned} k_G(\boldsymbol{x},\boldsymbol{x}') &= \frac{1}{(2\pi\sigma_{w_1}^2)^{n/2}}\int \exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{w}_{1i}|^2}{2\sigma_g^2} - \frac{|\boldsymbol{x}'-\boldsymbol{w}_{1i}|^2}{2\sigma_g^2} - \frac{\boldsymbol{w}_{1i}^T\boldsymbol{w}_{1i}}{2\sigma_{w_1}^2}\right)d\boldsymbol{w}_{1i} \\ &= \left(\frac{\sigma_e}{\sigma_{w_1}}\right)^n \exp\left(-\frac{\boldsymbol{x}^T\boldsymbol{x}}{2\sigma_m^2}\right)\exp\left(\frac{(\boldsymbol{x}-\boldsymbol{x}')^T(\boldsymbol{x}-\boldsymbol{x}')}{-2\sigma_s^2}\right)\exp\left(-\frac{\boldsymbol{x}'^T\boldsymbol{x}'}{2\sigma_m^2}\right), \end{aligned} \tag{2.92}$$

where $1/\sigma_e^2 = 2/\sigma_g^2 + 1/\sigma_{w_1}^2$, $\sigma_m^2 = 2\sigma_{w_1}^2 + \sigma_g^2$, and $\sigma_s^2 = 2\sigma_g^2 + \sigma_g^4/\sigma_{w_1}^2$. Both kernels k_{erf} and k_G are non-stationary, as a consequence of the Gaussian weight prior being centered on 0 which breaks translation invariance in weight space [4].

Kernel of Deep Neural Network

Since not every activation function has an analytical form for Equation 2.83, we only consider the activation that has a closed form, i.e. the ReLU. When no closed form exists, a numeric method described in [18] could be performed efficiently to realize a good approximation. For ReLU, the Equation 2.83 will be

$$K^l(\boldsymbol{x},\boldsymbol{x}') = \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\sqrt{K^{l-1}(\boldsymbol{x},\boldsymbol{x})K^{l-1}(\boldsymbol{x}',\boldsymbol{x}')}\left(\sin\theta_{x,x'}^{l-1} + (\pi - \theta_{x,x'}^{l-1})\cos\theta_{x,x'}^{l-1}\right), \tag{2.93}$$

$$\theta_{x,x'}^l = \cos^{-1}\left(\frac{K^l(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{K^l(\boldsymbol{x},\boldsymbol{x})K^l(\boldsymbol{x}',\boldsymbol{x}')}}\right). \tag{2.94}$$

With increasing depth of l, flattening angular structure of $K^l(\boldsymbol{x}, \boldsymbol{x}')$ is observed, and the numeric estimates coincide with the analytical results very well from authors' experiments.

Kernel of Convolutional Neural Network

As we discussed previously, we can always set the mean of a GP to 0, leading an emphasis only on covariance functions. Here the mean function is always 0 since we set biases and weights to have priors with zero means. In [13], covariance function of CNN for the 1st layer is:

$$\begin{aligned} K_\mu^{(1)}(\mathbf{X}, \mathbf{X}') &= \text{Cov}\left[A_{i\mu}^{(1)}(\mathbf{X}), A_{i\mu}^{(1)}(\mathbf{X}')\right] \\ &= \sigma_b^2 + \frac{\sigma_w^2}{C^{(0)}} \sum_{i=1}^{C^{(0)}} \sum_{\nu \in \mu\text{th path}} X_{i\nu} X'_{i\nu}, \end{aligned} \tag{2.95}$$

and for other layers

$$\begin{aligned} K_\mu^{(l+1)}(\mathbf{X}, \mathbf{X}') &= \text{Cov}\left[A_{i\mu}^{(l+1)}(\mathbf{X}), A_{i\mu}^{(l+1)}(\mathbf{X}')\right] \\ &= \sigma_b^2 + \sigma_w^2 \sum_{\nu \in \mu\text{th path}} V_\nu^{(l)}(\mathbf{X}, \mathbf{X}'), \end{aligned} \tag{2.96}$$

where

$$V_\nu^{(l)}(\mathbf{X}, \mathbf{X}') = \mathbb{E}\left[\phi\left(A_{j\nu}^{(l)}(\mathbf{X})\right)\phi\left(A_{j\nu}^{(l)}(\mathbf{X}')\right)\right] \tag{2.97}$$

is the covariance of activation functions, and ν and μ are locations within the input and output channels or feature maps. And we have already discussed the closed form solution $V_\nu^{(l)}$ for ReLU and error function in previous sections. We note that Equation 2.96 is a generalization of Equation 2.80 from single hidden layer networks to multilayer convolutional networks. Computing the kernel matrix takes $\mathcal{O}(N^2LD)$ time, where L is the number of layers, D is the dimensionality of input and N is the number of data points, and inverting the kernel matrix takes $\mathcal{O}(N^3)$. On MNIST, N^3 is around a factor of 10 larger than N^2LD [13]. In practice, it is usually more expensive to compute the kernel matrix than to convert it. Algorithm for computing the kernel $k(\mathbf{X}_1, \mathbf{X}_2)$ of two images $\mathbf{X}_1$ and $\mathbf{X}_2$ is shown in algorithm 1.

With a slight modification of adding skip connection between the activation of different layers, the Gaussianity of the network is still preserved due to the i.i.d activations, and we have a residual CNN as follows

$$\boldsymbol{\alpha}_i^{(l+1)}(\mathbf{X}) = \boldsymbol{\alpha}_i^{(l-s)}(\mathbf{X}) + \boldsymbol{b}_i^{(l+1)} + \sum_{j=1}^{C^{(l)}} \mathbf{W}_{ij}^{(l)} \phi\left(\boldsymbol{\alpha}_j^{(l)}(\mathbf{X})\right), \tag{2.98}$$

with its covariance

$$K_\mu^{(l+1)}(\mathbf{X}, \mathbf{X}') = K_\mu^{(l-s)}(\mathbf{X}, \mathbf{X}') + \sigma_b^2 + \sigma_w^2 \sum_{\nu \in \mu\text{th path}} V_\nu^{(l)}(\mathbf{X}, \mathbf{X}'), \tag{2.99}$$

Algorithm 1: Computing kernel of CNN-GPs

Input: two images $\mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^{C^{(0)} \times (H^{(0)} W^{(0)})}$

Output: the scalar $K_1^{(L+1)}(\mathbf{X}_1, \mathbf{X}_2)$

1 Compute $K_\mu^{(1)}(\mathbf{X}_1, \mathbf{X}_1)$, $K_\mu^{(1)}(\mathbf{X}_1, \mathbf{X}_2)$ and $K_\mu^{(1)}(\mathbf{X}_2, \mathbf{X}_2)$ for $\mu \in \{1, \ldots, H^{(1)} D^{(1)}\}$ using Equation 2.95;

2 **for** $l = 1, 2, \ldots, L$ **do**

3 | Compute $V_\mu^{(l)}(\mathbf{X}_1, \mathbf{X}_1)$, $V_\mu^{(l)}(\mathbf{X}_1, \mathbf{X}_2)$ and $V_\mu^{(l)}(\mathbf{X}_2, \mathbf{X}_2)$ for $\mu \in \{1, \ldots, H^{(l)} D^{(l)}\}$ using Equation 2.97;

4 | Compute $K_\mu^{(l+1)}(\mathbf{X}_1, \mathbf{X}_1)$, $K_\mu^{(l+1)}(\mathbf{X}_1, \mathbf{X}_2)$ and $K_\mu^{(l+1)}(\mathbf{X}_2, \mathbf{X}_2)$ for $\mu \in \{1, \ldots, H^{(l+1)} D^{(l+1)}\}$ using Equation 2.96.

5 **end**

where s is the number of skip connection spans.

Our later analysis of eigendecomposition and classification accuracy in Section 3 will base on kernels of residual CNN and CNN presented here.

3 Properties of Linear Operators Related to Gaussian Processes

Studying linear operators defined on kernels of neural networks is comparatively beneficial since this will reveal more information about how the geometric structure of original data changes after they are transformed by neural networks. Remember that a Gaussian process is determined by its kernel function for centered datasets and thus an integral operator can be applied on the kernel function to further understand the spectral properties of such a operator and how the data geometry changes.

In this section, we will first discuss more technical aspects concerning the integral operator related to GP kernels and the spectral properties of the operator. After that, we will mainly pay our attention to the numerical experiments on eigen-analysis of kernel matrices induced by convolutional neural networks as Gaussian processes (CNN-GPs) and residual convolutional neural networks as Gaussian processes (ResCNN-GPs) such that geometric changes of data can be observed under different network architectures resulted from changing network depths and varying kernel sizes resulted from differing numbers of input images, and finally we inspect the influence of kernel matrices on classification accuracy for both cases.

Our experiments are conducted on MNIST data set, which is the classic handwritten digits originally used to train LeNet-5 proposed by LeCun et al. [17] in 1989. MNIST data set contains 60,000 training images and 10,000 test images. Every handwritten digit is cropped to a size of 28×28 pixels with only one channel, and each pixel is scaled in the range $[0, 255]$. For the computation of kernel matrices, each pixel must be normalized to $[0, 1]$, for which each pixel just needs to be divided by 255 to be normalized in such a range. Figure 3.1 shows what the MNIST data set looks like, and the above are their corresponding labels.

3.1 Hilbert-Schmidt Integral Operator

We can see from Section 2, there are various linear operators available, we will focus on a linear operator that is always applied for kernel functions, which is the operator T_k we mentioned in Section 2.2.2.

3.1.1 The Integral Operator for Kernels

The linear operator related to kernels has the form:

$$[T_k\phi](\boldsymbol{x}) = \int k(\boldsymbol{x}, \boldsymbol{x}')\phi(\boldsymbol{x}')d\boldsymbol{x}', \tag{3.1}$$

which is also known as Hilbert-Schmidt integral operator. For a finite situation, we have:

$$[T_k\phi](\boldsymbol{x}) = \sum_{i=1}^{n} k(\boldsymbol{x}, \boldsymbol{t}_i)\phi(\boldsymbol{t}_i). \tag{3.2}$$

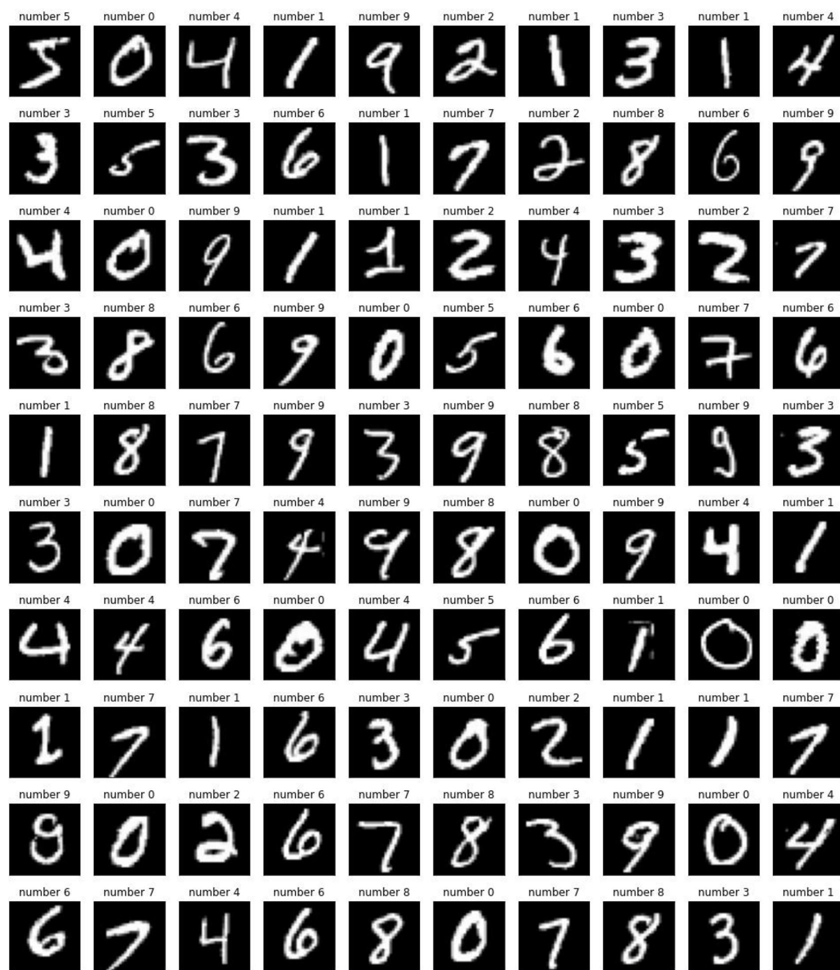


Figure 3.1: MNIST data with labels above for each image.

We know from Mercer's theorem in Section 2.2.2 that:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{+\infty} \lambda_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}') = \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}') \rangle , \tag{3.3}$$

where $\boldsymbol{\psi}(\cdot)$ maps $\boldsymbol{x}$ and $\boldsymbol{x}'$ to another infinite dimensional vector space, which is represented by

$$\boldsymbol{\psi}(\cdot) = \left(\sqrt{\lambda_1}\phi_1(\cdot), \ldots, \sqrt{\lambda_n}\phi_n(\cdot), \ldots \right)^T . \tag{3.4}$$

Substituting Equation 3.3 into the integral operator Equation 3.1 above, we have the following:

$$\begin{aligned}
[T_k \phi_i](\boldsymbol{x}) &= \int k(\boldsymbol{x}, \boldsymbol{x}') \phi_i(\boldsymbol{x}') d\boldsymbol{x}' \\
&= \int \sum_{j=1}^{+\infty} \lambda_j \phi_j(\boldsymbol{x}) \phi_j(\boldsymbol{x}') \phi_i(\boldsymbol{x}') d\boldsymbol{x}' \\
&= \sum_{j=1}^{\infty} \lambda_j \phi_j(\boldsymbol{x}) \int \phi_j(\boldsymbol{x}') \phi_i(\boldsymbol{x}') d\boldsymbol{x}' \\
&= \sum_{j=1}^{\infty} \lambda_j \phi_j(\boldsymbol{x}) \langle \phi_j, \phi_i \rangle \\
&= \lambda_i \phi_i(\boldsymbol{x}),
\end{aligned} \tag{3.5}$$

where λ_i and ϕ_i are eigenvalues and eigenfunctions of T_k operator respectively, and we use the orthogonality of basis functions ϕ_i that $\langle \phi_i, \phi_j \rangle = 1$ if $i = j$ and $\langle \phi_i, \phi_j \rangle = 0$ otherwise.

3.1.2 Properties of the Integral Operator

Recall from previous section of linear operators that linear operators can be represented by a matrix on a finite number of basis functions, and with the help of Mercer's theorem we can easily formulate the matrix representing operator T_k since the operator is closed in the subspace spanned by n bases $k(\boldsymbol{x}_i, \cdot)$ for $i = 1, \ldots, n$, which indicates that the operator T_k can be represented by the basis functions if we apply T_k to the bases. The matrix is denoted by $\mathbf{K}$, which consists of kernel functions and has the form as follows:

$$\mathbf{K} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix} \in \mathbb{R}^{n \times n},$$

and this matrix $\mathbf{K}$ is exactly what we call Gram matrix. When we apply same orthogonal basis function $\phi_i(\cdot)$ on data points $\boldsymbol{x}_i, i = 1, \ldots, n$ for each row of the Gram matrix $\mathbf{K}$,

we have the eigendecomposition of Gram matrix $\mathbf{K}$ as follows:

$$\begin{aligned}\mathbf{K}\boldsymbol{\phi}_i &= \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix} \begin{pmatrix} \phi_i(\boldsymbol{x}_1) \\ \phi_i(\boldsymbol{x}_2) \\ \phi_i(\boldsymbol{x}_3) \\ \vdots \\ \phi_i(\boldsymbol{x}_n) \end{pmatrix}, \\ &= \lambda_i \begin{pmatrix} \phi_i(\boldsymbol{x}_1) \\ \phi_i(\boldsymbol{x}_2) \\ \phi_i(\boldsymbol{x}_3) \\ \vdots \\ \phi_i(\boldsymbol{x}_n) \end{pmatrix}\end{aligned} \tag{3.6}$$

where $\boldsymbol{\phi}_i \in \mathbb{R}^n$ for $i = 1, \dots, n$ takes the form:

$$\boldsymbol{\phi}_i = \begin{pmatrix} \phi_i(\boldsymbol{x}_1) \\ \phi_i(\boldsymbol{x}_2) \\ \phi_i(\boldsymbol{x}_3) \\ \vdots \\ \phi_i(\boldsymbol{x}_n) \end{pmatrix} \in \mathbb{R}^n.$$

From above, we have the eigenvector matrix of Gram matrix of kernels:

$$\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_n),$$

and eigenvalue matrix is:

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}, \tag{3.7}$$

where we permute the order of eigenvalues as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ along the diagonal and set other entries to be 0, we can thus write the kernel matrix as:

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^{\mathrm{T}}, \tag{3.8}$$

which has an exact representation as PCA. The set of eigenvalues for operator T_k is called its spectrum.

Here we only focus on the integral operator T_k on kernels. Consequently, if we are going to analyze the integral operator of kernel functions, it is sufficient for us to conduct eigenanalysis for the Gram matrix of kernels, which shares identical eigenvalues and corresponding eigenvectors with the integral operator. Therefore, eigenvalues and eigenvectors are the objects we pay most of the attention to, and our analysis on Section 3.2 and Section 3.3 will pertain to the eigendecomposition of Gram matrix of kernels (which also refers to kernel matrix or covariance matrix in later sections).

3.1.3 An Example Based on Radial Basis Function

For the clear comprehension of how the previously mentioned operator conduct on kernels, we use radial basis function kernel (RBF, also called Gaussian kernel) as an instance. RBF takes the form:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{d(\boldsymbol{x}, \boldsymbol{x}')^2}{2l^2}\right), \tag{3.9}$$

where $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$, $d(\cdot, \cdot)$ is the Euclidean distance and l is the length scale. In order to find the eigenfunctions and eigenvalues of the RBF, we set $l = 1$ and take $x, y \in \mathbb{R}$ for simplicity and thus Equation 3.9 is rewritten as follows:

$$\begin{aligned}
k(x, y) &= \exp\left(-\frac{(x-y)^2}{2}\right) \\
&= \exp\left(-\frac{x^2+y^2-2xy}{2}\right) \\
&= \exp\left(-\frac{x^2+y^2}{2}\right)\exp(xy) \\
&= \exp\left(-\frac{x^2}{2}\right)\exp\left(-\frac{y^2}{2}\right)\left(1 + \frac{xy}{1!} + \frac{(xy)^2}{2!} + \cdots + \frac{(xy)^\infty}{\infty!}\right) \\
&= \underbrace{\exp\left(-\frac{x^2}{2}\right)\left(1, \sqrt{\frac{x}{1!}}, \sqrt{\frac{x^2}{2!}}, \cdots, \sqrt{\frac{x^\infty}{\infty!}}\right)}_{=\boldsymbol{\psi}(x)} \underbrace{\exp\left(-\frac{y^2}{2}\right)\left(1, \sqrt{\frac{y}{1!}}, \sqrt{\frac{y^2}{2!}}, \cdots, \sqrt{\frac{y^\infty}{\infty!}}\right)}_{=\boldsymbol{\psi}(y)},
\end{aligned} \tag{3.10}$$

where we first use Taylor expansion for the term $\exp(xy)$ and then rewrite the expansion in the form of inner product. Comparing Equation 3.3 and Equation 3.10, we therefore have the explicit forms of eigenvalues λ_i and eigenfunctions ϕ_i for $i = 1, \ldots, \infty$:

$$\lambda_i = \sqrt{\frac{1}{(i-1)!}}, \tag{3.11}$$

$$\phi_i(x) = \sqrt{x^{i-1}}\exp\left(-\frac{x^2}{2}\right), \tag{3.12}$$

respectively, after applying operator T_k on RBF. When i becomes larger, eigenvalues λ_i will converge to 0 and eigenfunctions ϕ_i will fluctuate for differing x. Clearly, RBF kernel maps 1D data points into infinite-dimensional vectors in a new space. Equation 3.10 can be easily extended to multidimensional cases as well. For the kernel functions that are difficult to find the explicit formulas for eigenvalues and eigenfunctions, the approximation methods in Section 2.2.2 are relatively helpful in finding viable solutions.

3.2 Networks with Different Architectures

For the experiments, we analyze the eigendecomposition of CNN-GPs and ResCNN-GPs on the same batch of 500 data points with respect to architectures, respectively. In [13],

the authors used a 7-layer CNN with 7×7 filters and padded zeros for each layer to keep the size identical and each convolutional layer is followed by ReLU nonlinearity. In addition, all the networks have a fully connected layer for the final layer with a filter of size 28×28, which outputs a single value. For instance, we define CNN-GPs and ResCNN-GPs as in Figure 3.2. Here the library **cnn_gp** is from the modification work of Tim Waegemanns [31] based on that of Carl E. Rasmussen et. al. [13], which is more numerically stable. Note that the variances σ_b, σ_w, number of layers, stride, filter size, the nonlinearity ϕ, and the skip connection span s are all hyperparameters that influence the kernel matrices. We will base our analysis on networks with optimized hyperparameters and only alter the depth of our networks, while the other hyperparameters are fixed for networks. Related codes for the experiments are available under `https://github.com/StevenXuf/Spectral_properties_of_kernels_of_CNN-GPs`.

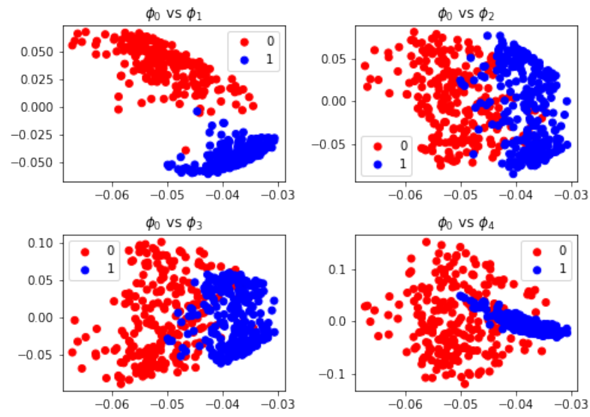
3.2.1 Eigen-analysis of Kernel Matrices

We do eigendecomposition for 3 CNN-GPs and 3 ResCNN-GPs, respectively, with different numbers of layers, for which only images of 2 digits are used as an input for our networks at first. In order to have apparent comparisons, we analyze most possibly confused digits 3 and 8, and the most visually distinctive digits 0 and 1. First, we analyze the top 5 eigenvectors for 3 varying depths for CNN-GPs, as shown in Figure 3.3. Figure 3.3a plots the top 5 eigenvectors for CNN-GP with 3 layers (CNN-GP3), and $\boldsymbol{\phi}_0$ is the first eigenvector corresponding to the largest eigenvalue and is used for x-axis, while $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_5$ are eigenvectors corresponding to the 2nd largest, ..., 5th largest eigenvalue, and used as y-axis respectively. Figure 3.3b and Figure 3.3c plot similar pairwise eigenvectors with only disparity of differing layers for CNN-GP5 and CNN-GP7. Note that every pair of eigenvectors $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$ are orthogonal and thus $\boldsymbol{\phi}_i^T \boldsymbol{\phi}_j = 0$ holds if $i \neq j$. Form the pair plots of CNN-GP for digit 0 and 1, we see that:

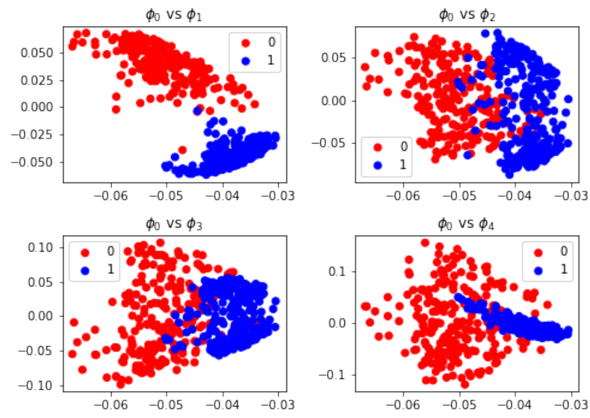
- each pair plot of all networks is clearly clustered and the points for digit 1 is more tightly clustered than points of digit 0;
- first two eigenvectors can separate digit 0 and 1 very well;
- the first eigenvector $\boldsymbol{\phi}_0$ of all 3 CNN-GPs with differing depths has nearly the same upper bound and lower bound, while the bounds of other 4 eigenvectors differ;
- for the pair plot of a network, the range length of eigenvectors tends to increase as we see from the y-axis, i.e. $L(\text{range}(\phi_0)) \leq L(\text{range}(\phi_1)) \leq L(\text{range}(\phi_2)) \leq L(\text{range}(\phi_3)) \leq L(\text{range}(\phi_4))$. And this holds for all 3 CNN-GPs;
- the pair plots of a network are similar to those of another network, respectively, no matter which two networks we choose;
- the 4th pair plot of CNN-GP5 and that of CNN-GP7 is visually turned upside down;

```
from cnn_gp import Sequential,ReLU,Conv2d,Sum

def cnn(n_layers,filter_size=7,var_b=7.86,var_w=2.79,stride=1):
    layers=[]
    for i in range(n_layers):
      layers+=[Conv2d(kernel_size=filter_size,
                      padding='same',
                      var_weight=var_w*filter_size**2,
                      var_bias=var_b,
                      stride=stride),
               ReLU()]
    model=Sequantial(*layers,Conv2d(kernel_size=28,
                                   padding=0,
                                   var_weight=var_w,
                                   var_bias=var_b))
    return model

def res_cnn(n_layers,filter_size=4,var_b=4.69,var_w=7.27,stride=1):
    model=Sequential(
          *(Sum([Sequential(),
                 Sequential(
                 Conv2d(kernel_size=filter_size,
                        padding='same',
                        var_weight=var_w*filter_size**2,
                        var_bias=var_b,
                        stride=stride),
                 ReLU())
               ]) for i in range(n_layers-1)),
          Conv2d(kernel_size=filter_size,
                 padding='same',
                 var_weight=var_w*filter_size**2,
                 var_bias=var_b,
                 stride=stride),
          ReLU(),
          Conv2d(kernel_size=28,
                 padding=0,
                 var_weight=var_w,
                 var_bias=var_b)
          )
    return model
```

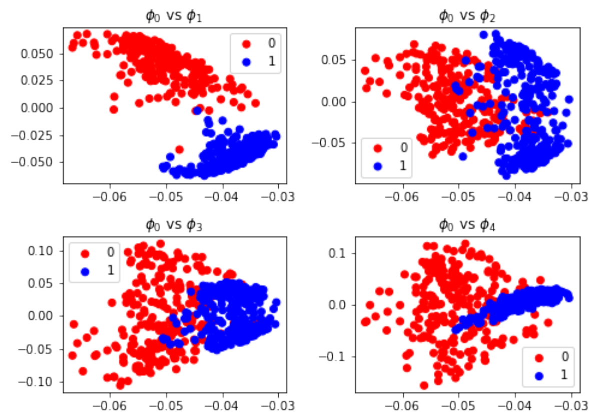
Figure 3.2: Codes of CNN-GPs and ResCNN-GPs.



(a) CNN-GP3.



(b) CNN-GP5.



(c) CNN-GP7.

Figure 3.3: Pairwise comparisons of top 5 eigenvectors for 0 and 1 under CNN-GPs.

- values of the first eigenvector $\boldsymbol{\phi}_0$ are always negative, while the values of the other 4 eigenvectors can be positive or negative.

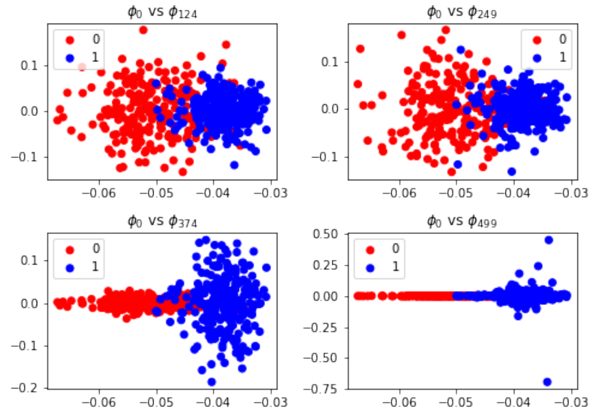
We do not see a significant influence on eigenvectors from CNN-GPs with different numbers of layers except for some small changes of eigenvectors, which implies that the number of layers does affect the kernels, but has little influence on the eigenvectors of kernel matrices. This further implicates that the eigenvectors are much decided by intrinsic features of data itself and the integral operator. Figure 3.4 shows more eigenvectors of kernel matrices so that we can grasp the general global trends, and we observe convergence for eigenvectors, especially for those of digit 0. Same patterns as those of CNN-GPs are observed for the eigenvectors of ResCNN-GPs with 3, 5, 7 layers in Figure 3.5 and Figure 3.6. And there is no notable differences observed from CNN-GPs and ResCNN-GPs.

To further instantiate whether the depth of a network has impacts on eigenvectors, we plot 3D scatter plots of top 3 eigenvectors for kernel matrices induced by CNN-GPs in Figure 3.7, and ResCNN-GPs in Figure 3.8, respectively, on images consisting of digits 0 and 1. From both figures, we draw a conclusion that eigenvectors of kernel matrices are not significantly influenced by network depths, regardless of CNN-GPs or ResCNN-GPs, and eigenvectors of digit 0 and digit 1 have a clear boundary between 2 clusters.

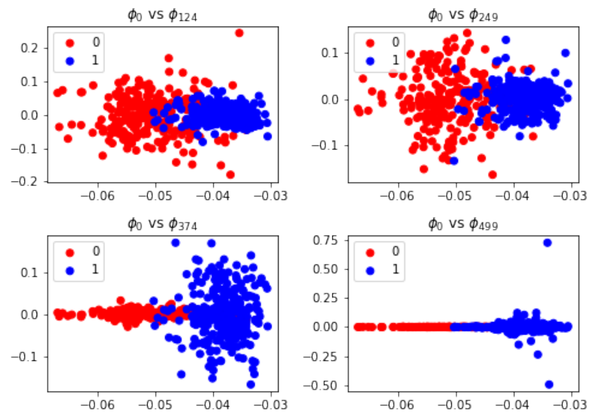
We also examine the behaviors of 10 chosen eigenvectors in Figure 3.9 to see how the dispersion and skewness of eigenfunctions ϕ_i's changes, from which we see that eigenfunctions ϕ_i are bounded and tended to convergence generally. The reason for bounded eigenfunctions is because eigenfunctions ϕ_i's are determined by the integral operator T_k and we have a limit data points forming a submanifold in space, which causes bounded eigenfunctions. We also observe that values of eigenvectors are symmetrically spread around 0, indicating no significant skewness. Furthermore, no distinctive differences among CNN-GPs with varying depths are founded.

After analyzing the eigenvectors, we then plot the logarithms of eigenvalues for CNN-GPs and ResCNN-GPs with different depths, respectively, as shown in Figure 3.10. For CNN-GPs, we observe that the decreasing trends for 3 CNN-GPs are very similar and the eigenvalues of the kernel matrix induced by the network with more layers will have larger eigenvalues. In the beginning, the eigenvalues drop fast and then decrease slowly, indicating that the first several eigenvectors are more important than others. The same phenomenon holds for ResCNN-GPs. More interestingly, if we compare CNN-GP and ResCNN-GP with the same depths, we see from that with more layers the curve of eigenvalues of CNN-GP and ResCNN-GP are tended to coincide with each other.

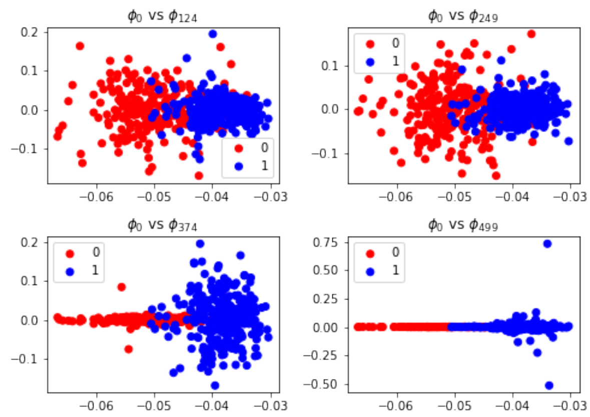
Next we perform same analysis for CNN-GPs on images of digits 3 and 8 in Figure 3.11 and Figure 3.12. Again, the network depths seem do not significantly influence the eigenvectors of related kernel matrices. One especially important distinction with figures of digits 0 and 1 is that the pairwise plots of eigenvectors of kernel matrices on images of digits 3 and 8 are tangled too much, which might arise from the visual confusion caused by handwriting. Pairwise plots of eigenvectors of kernel matrices induced by ResCNN-GPs are shown in Figure 3.13 and Figure 3.14. And the skip connection of ResCNN-GPs do not bring us too many visually distinguishable differences, which



(a) CNN-GP3.

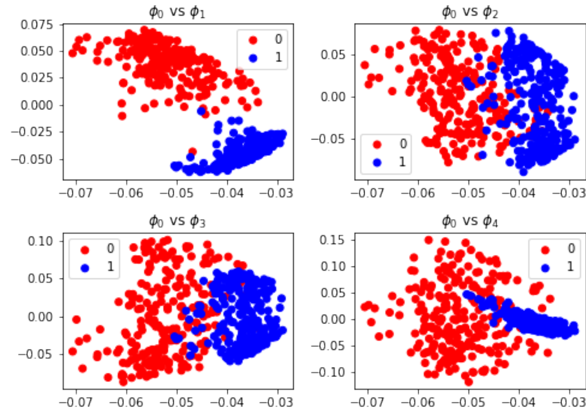


(b) CNN-GP5.

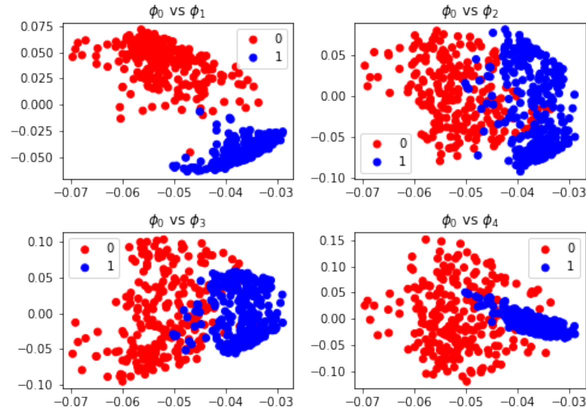


(c) CNN-GP7.

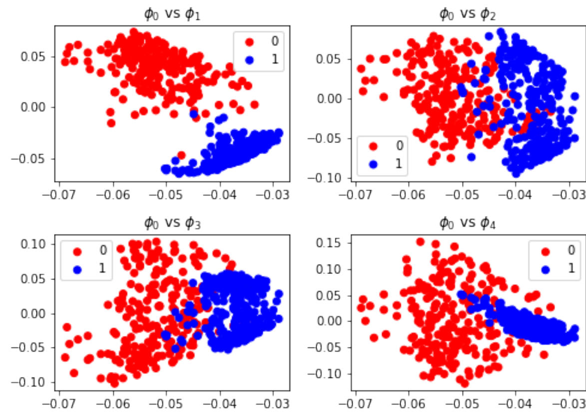
Figure 3.4: Pairwise comparisons of more eigenvectors for 0 and 1 under CNN-GPs.



(a) ResCNN-GP3.

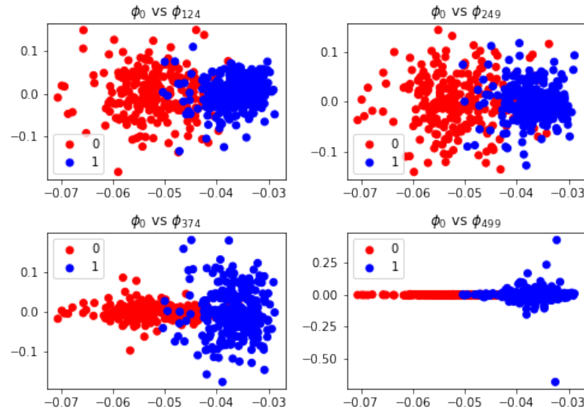


(b) ResCNN-GP5.

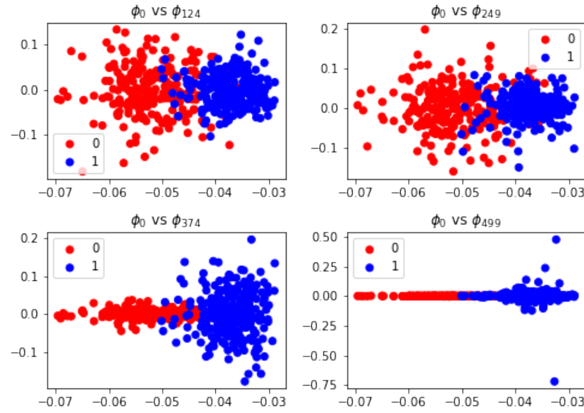


(c) ResCNN-GP7.

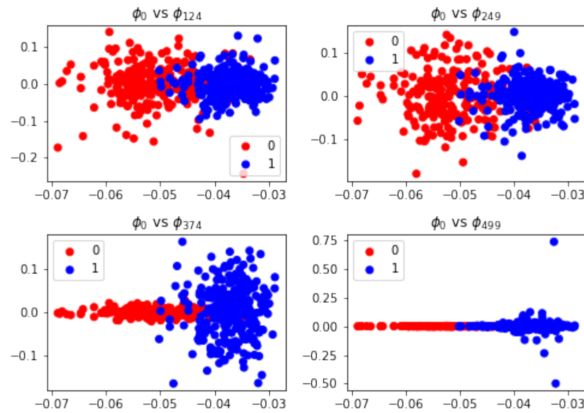
Figure 3.5: Pairwise comparisons of top 5 eigenvectors of kernel matrices induced by CNN-GPs for digits 0 and 1.



(a) ResCNN-GP3.

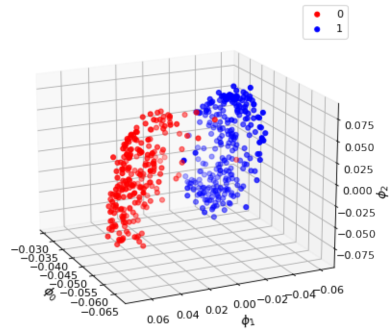


(b) ResCNN-GP5.

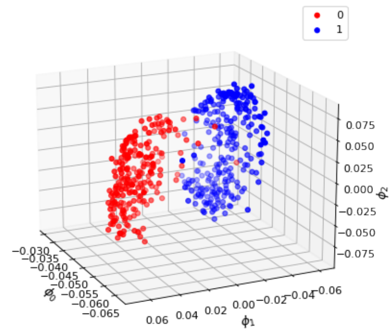


(c) ResCNN-GP7.

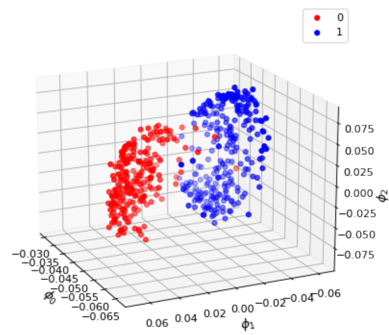
Figure 3.6: Pairwise comparisons of more eigenvectors of kernel matrices induced by CNN-GPs for digits 0 and 1.



(a) CNN-GP3

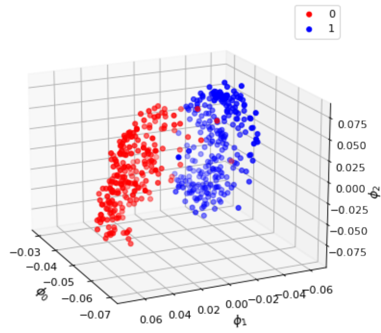


(b) CNN-GP5.

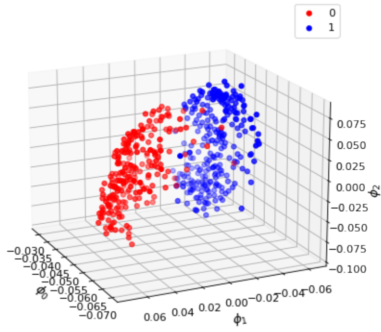


(c) CNN-GP7.

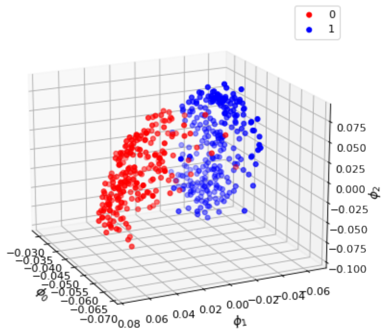
Figure 3.7: Scatter plots of top 3 eigenvectors of kernel matrices induced by CNN-GPs on digits 0 and 1.



(a) ResCNN-GP3

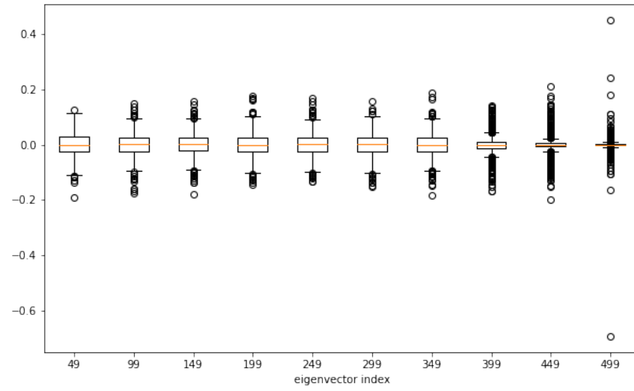


(b) ResCNN-GP5.

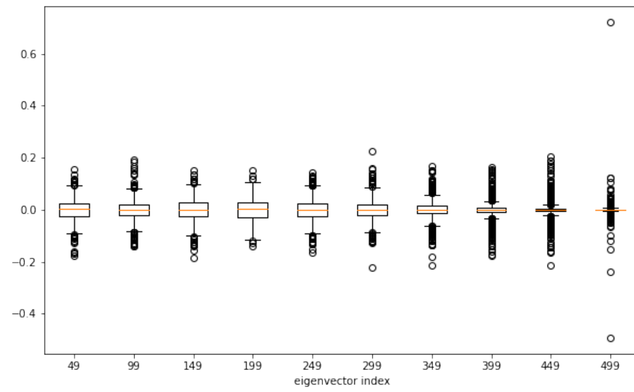


(c) ResCNN-GP7.

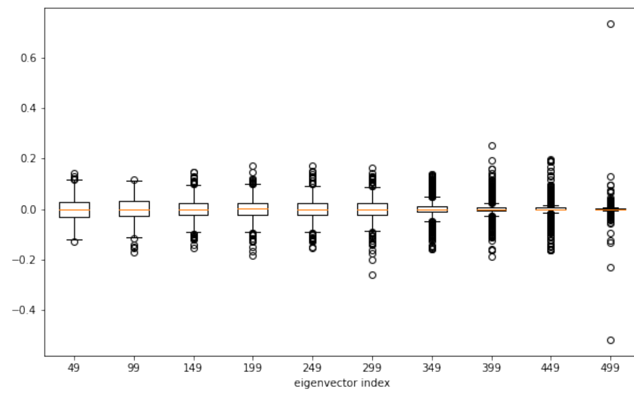
Figure 3.8: Scatter plots of top 3 eigenvectors of kernel matrices induced by ResCNN-GPs on digits 0 and 1.



(a) Boxplot of CNN-GP3.



(b) Boxplot of CNN-GP5.



(c) Boxplot of CNN-GP7.

Figure 3.9: Boxplots of 10 chosen eigenvectors of kernel matrices induced by CNN-GPs on images of digits 0 and 1.

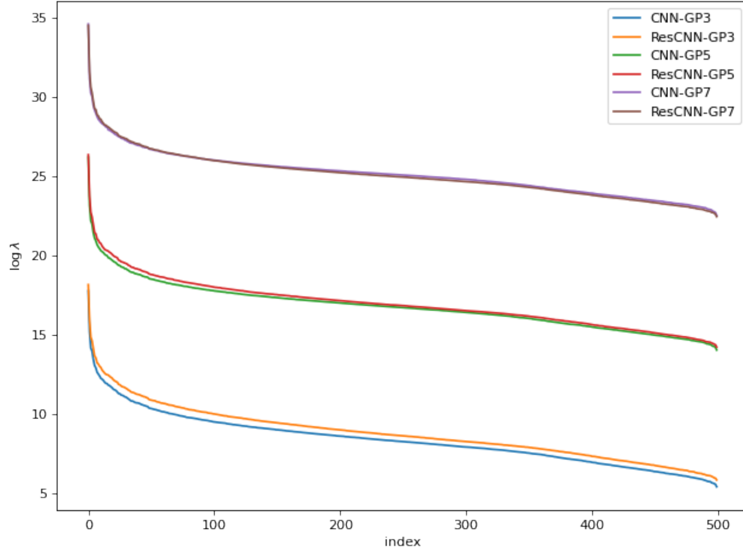
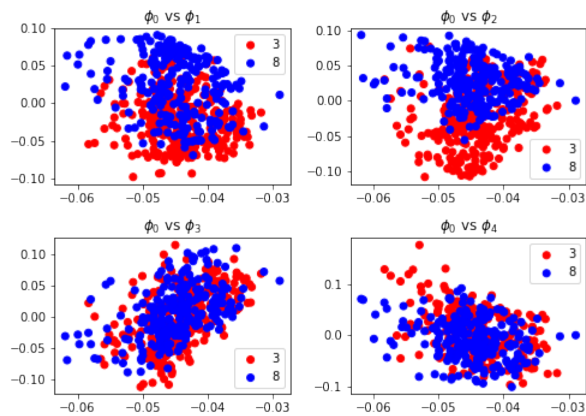


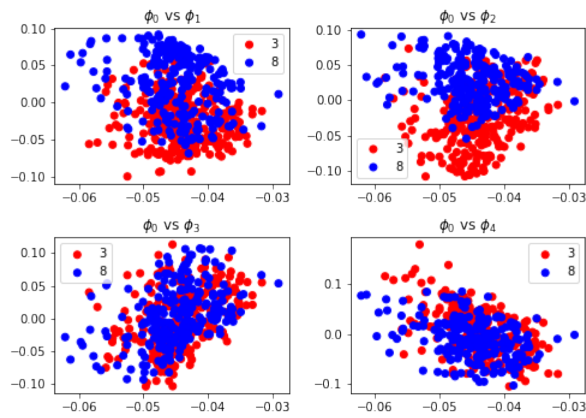
Figure 3.10: Eigenvalues of networks with different depths for digits 0 and 1.

possibly results from the added kernel $K_{\mu}^{l-s}(\mathbf{X},\mathbf{X})$ of precedent layers on current kernel $K_{\mu}^{l}(\mathbf{X},\mathbf{X})$ in Equation 2.99. That is to say, essentially the kernels of ResCNN-GPs are just multiple additions of kernel matrices of CNN-GPs from preceding layers. 3D plots for CNN-GPs and ResCNN-GPs on images of digit 3 and digit 8 are visualized in Figure 3.15 and Figure 3.16, respectively. From both figures, we draw a similar conclusion as the 3D scatter plots of Figure 3.3 and Figure 3.5 except that there is no clear boundary between eigenvectors for digit 3 and digit 8, which results from the confusability of these two handwritten digits, regardless CNN-GPs or ResCNN-GPs.

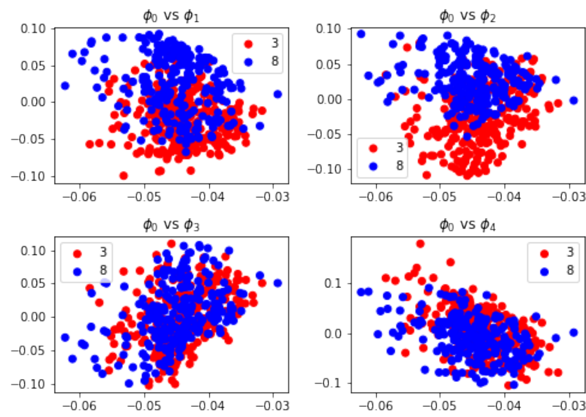
The logarithms of eigenvalues of networks to digits 3 and 8 are shown in Figure 3.17. Once again, the eigenvectors of CNN-GPs and ResCNN-GPs gave similar decreasing trends. Although the curves of CNN-GPs and ResCNN-GPs with the same number of layers tend to coincide with each other, curves of eigenvectors of ResCNN-GPs have a lower tail that seems to diverge from that of CNN-GPs. Further analyses are conducted on kernel matrices induced by CNN-GP3 on datasets consisting of 800 images of digits 3, 5, and 8 and 1000 images of digits 1, 3, 5, and 8, respectively. Analysis for more digits or more eigenvectors would be unnecessary since from the aforementioned experiments we have picked the most possibly visual distinguishable digits and the most confusable digits from all ten digits and the convergence trend of eigenvectors shall be similar, and we speculate that other cases should fall in between two extremes. And the skip connection as well as network depth have no significant impacts on eigenvectors, we hence only focus on CNN-GP3. Related results are shown in Figure 3.18. It is not surprising that the eigenvectors corresponding to each digit are highly tangled, no matter in 2D or 3D Euclidean space, which pertains to the indistinguishable shapes (or similarities to a certain degree) of 3 and 5 written by hands. We also add digit 1 to see whether there



(a) CNN-GP3.

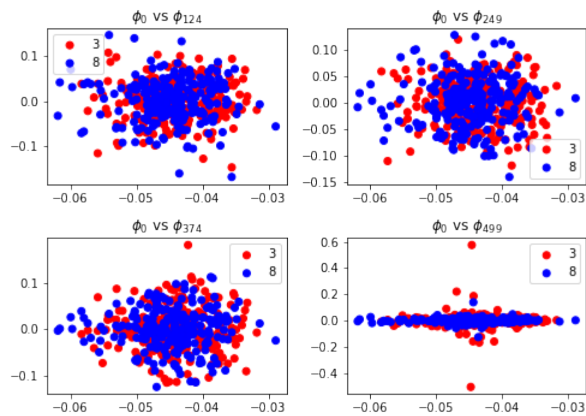


(b) CNN-GP5.

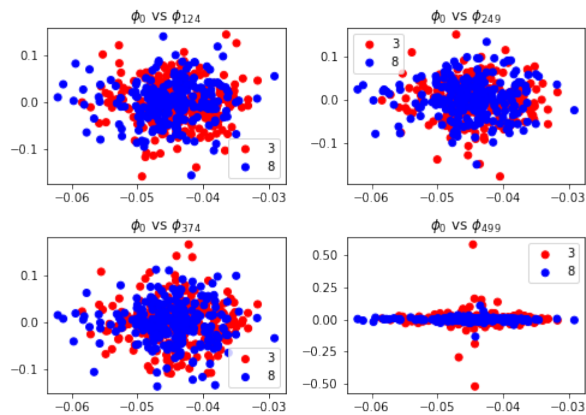


(c) CNN-GP7.

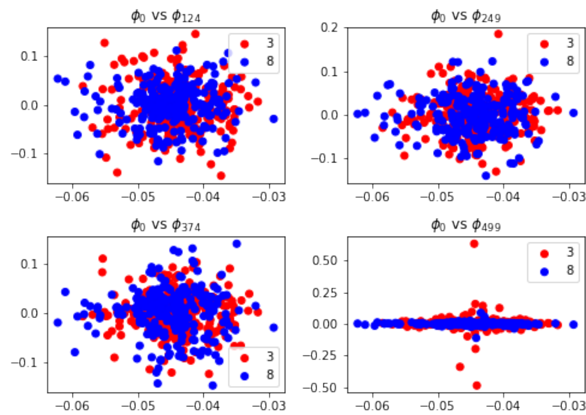
Figure 3.11: Pairwise comparisons of top 5 eigenvectors of kernel matrices induced by CNN-GPs for digits 3 and 8.



(a) CNN-GP3.

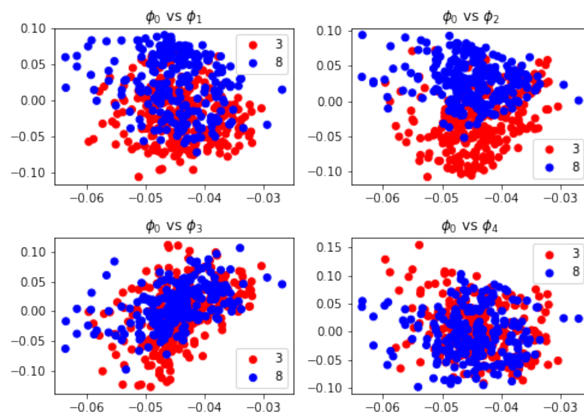


(b) CNN-GP5.

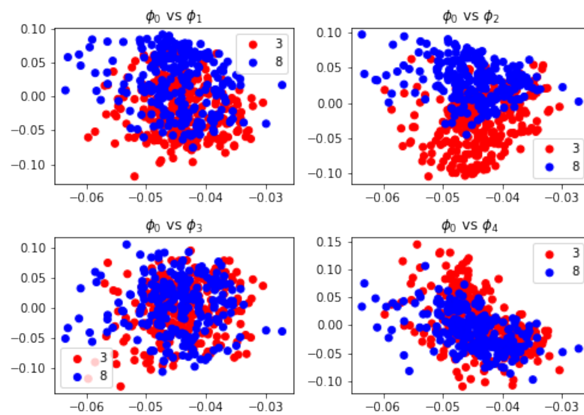


(c) CNN-GP7.

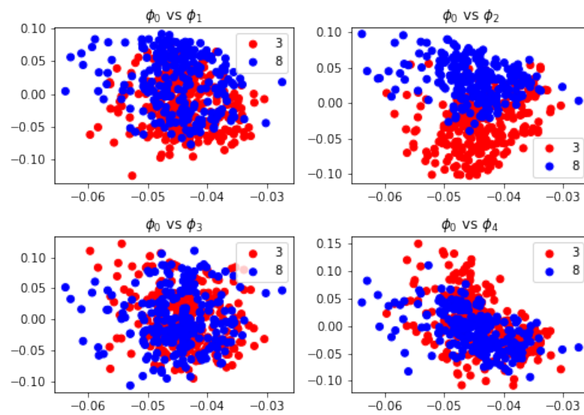
Figure 3.12: Pairwise comparisons of more eigenvectors of kernel matrices induced by CNN-GPs for digits 3 and 8.



(a) ResCNN-GP3.

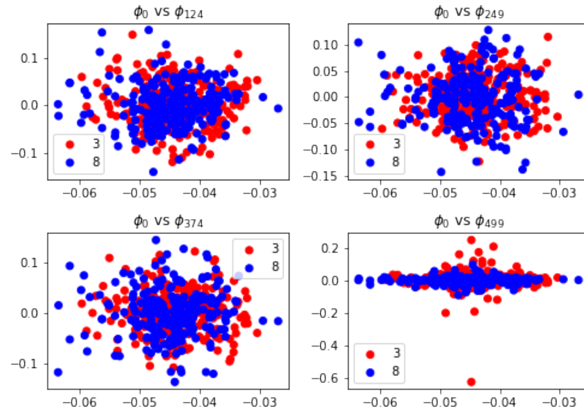


(b) ResCNN-GP5.

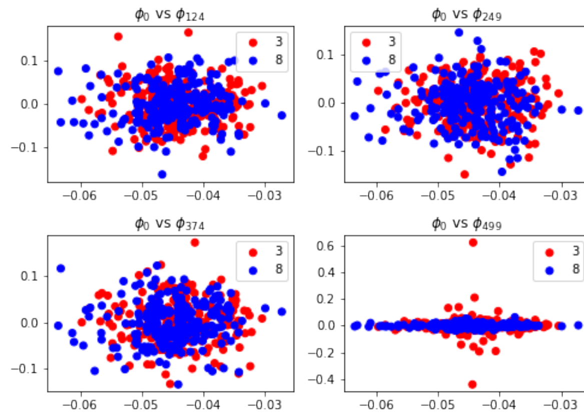


(c) ResCNN-GP7.

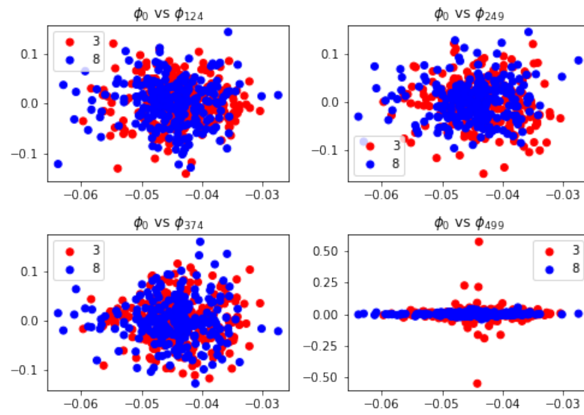
Figure 3.13: Pairwise comparison of top 5 eigenvectors of kernel matrices induced by ResCNN-GPs for 3 and 8.



(a) ResCNN-GP3.

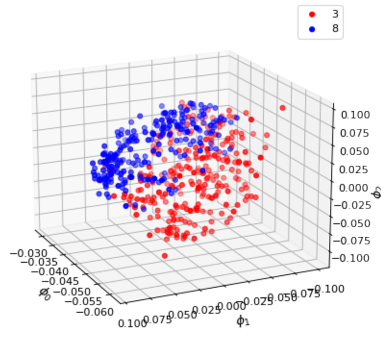


(b) ResCNN-GP5.

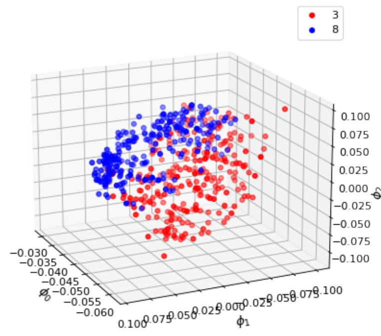


(c) ResCNN-GP7.

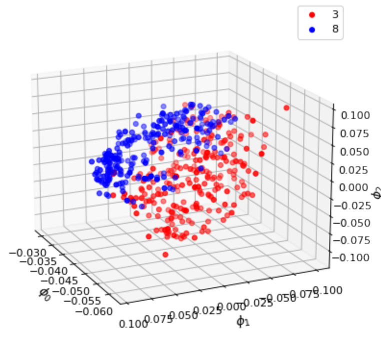
Figure 3.14: Pairwise comparison of more eigenvectors of kernel matrices induced by ResCNN-GPs for 3 and 8.



(a) CNN-GP3

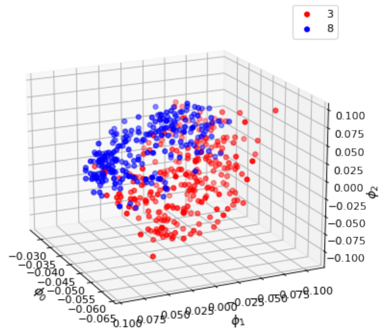


(b) CNN-GP5.

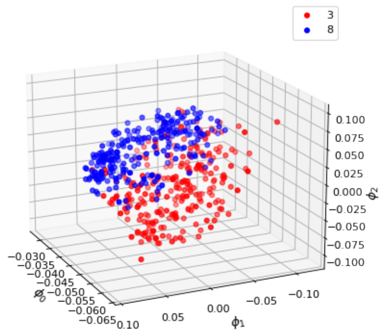


(c) CNN-GP7.

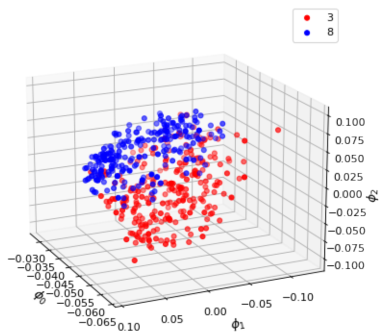
Figure 3.15: Scatter plots of top 3 eigenvectors of kernel matrices induced by CNN-GPs on digits 3 and 8.



(a) ResCNN-GP3



(b) ResCNN-GP5.



(c) ResCNN-GP7.

Figure 3.16: Scatter plots of top 3 eigenvectors of kernel matrices induced by ResCNN-GPs on digits 3 and 8.

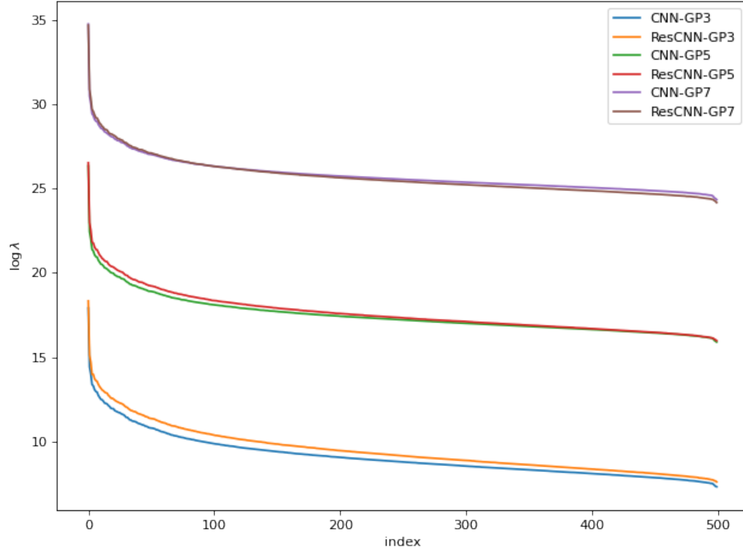


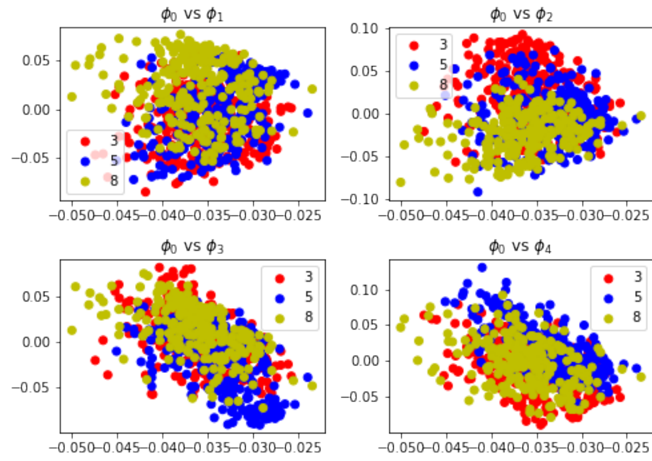
Figure 3.17: Eigenvalues of networks with different depths for digits 3 and 8.

is a distinctive gap or boundary between 1 and the rest in Figure 3.19. It is very clear that eigenvectors corresponding to digit 1 is not much tangled with the rest in 2D, and the gap is even more legible in 3D space, which seems be very reasonable since digit 1 can be easily told from the rest by vision. Theoretically, if we think about two point $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$ that are very close to each other, then for a eigenfunction $\phi_i(\cdot) : \mathbb{R}^n \to \mathbb{R}$ we have corresponding function values $\phi_i(\boldsymbol{x}_1)$ and $\phi_i(\boldsymbol{x}_2)$ that are in a vicinity as well, which is exactly the Lipschitz continuity. Figure 3.20 gives an illustration.

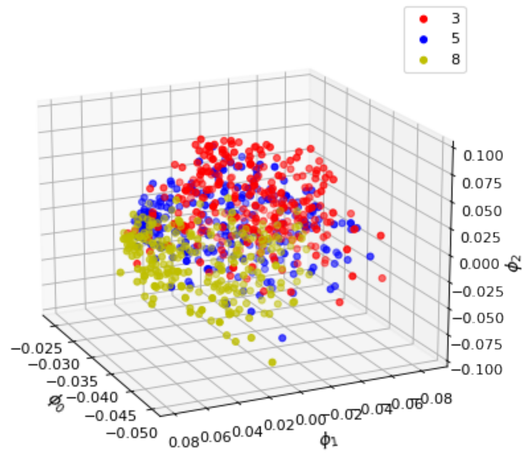
From experiments concerning eigenvectors and eigenvalues of kernel matrices induced by CNN-GPs and ResCNN-GPs, respectively, we conclude that the depth of a network does not play a significant role for the variations of eigenvectors but for eigenvalues, CNN-GPs and ResCNN-GPs do not make many differences, which verifies that the skip connection has only diminutive influences on the eigenanalysis of kernel matrices, and the eigenvectors are much affected by the intrinsic similarity of data themselves.

3.2.2 Performance on Classification

In classic neural networks, with deeper layers the accuracy of a network will increase until reaching a certain depth. However, will CNN-GPs and ResCNN-GPs have the same behavior as classic networks? If it is not the case that more layers ensure higher accuracy, then what is the best depth for each network? Here we analyze how the networks with differing depths influence the performance of classification accuracy on MNIST dataset. For the computation of classification accuracy, we first split the whole training data into 120 batches and split the whole test data into 20 batches. We will pick 3 random training batches $\mathbf{X}_j, 1 \leq j \leq 100$ without replacement to compute $\mathbf{K}(\mathbf{X}_j, \mathbf{X}_j)$

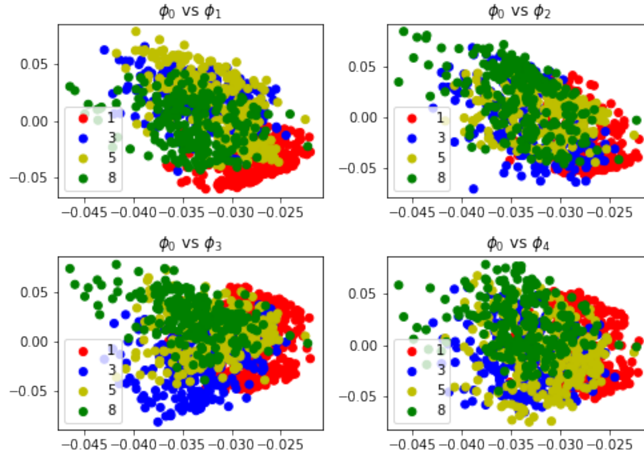


(a) Pairwise plots of top 5 eigenvectors.

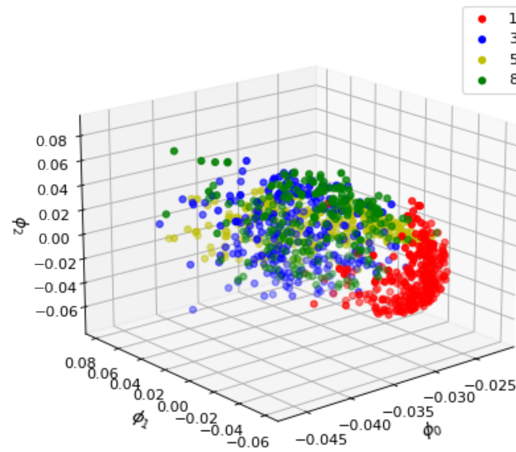


(b) Scatter plot of top 3 eigenvectors.

Figure 3.18: Eigenvectors of the kernel matrix induced by CNN-GP3 on the dataset consisting 800 images of digits 3, 5, and 8 in total.



(a) Pairwise plots of top 5 eigenvectors.



(b) Scatter plot of top 3 eigenvectors.

Figure 3.19: Eigenvectors of the kernel matrix induced by CNN-GP3 on the dataset consisting 1000 images of digits 1, 3, 5, and 8 in total.

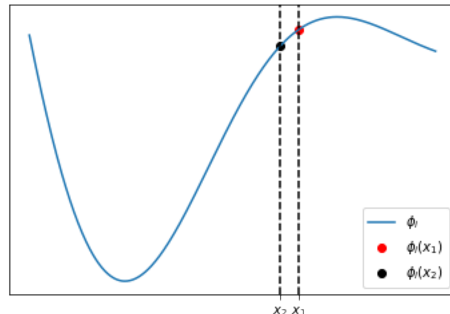


Figure 3.20: Function values of 2 close data points. Two close (similar) images $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ should have close (similar) values for same eigenfunction ϕ_i.

and run all the test batches $\mathbf{X}_i^*, i = 1, \ldots, 20$ in sequence for each chosen training batch, leading to 20 $\mathbf{K}(\mathbf{X}_i^*, \mathbf{X}_j), i = 1, \ldots, 20$, where $\mathbf{X}_i^*$ and $\mathbf{X}_j$ are normalized tensor of size $(500, 1, 28, 28)$. We have shuffled both the training and test data before training our networks, which avoids sampling biases. The 3 chosen random training data for all CNN-GPs and ResCNN-GPs are the same.

We first compute the number of total correctly predicted data points on the whole test data set for 5 CNN-GPs with the only difference of network depths (other parameters are fixed), and calculate their respective average accuracy and standard deviations of correctly predicted class labels over 20 test runs for each training batch, as shown in Table 3.1. We run each model on 3 randomly chosen training batches and predict on test batch of size 500 each time until all the test points are used. The average accuracy is computed on all 3 training runs for each model. Obviously, CNN-GP5 is better than CNN-GP3, and CNN-GP7 is better than CNN-GP5. Accuracy drops from CNN-GP7, and CNN-GP7 has an overall better performance than others, regardless of average accuracy or standard deviation. A very surprising discovery is that the network CNN-GP11 has a significantly terrible performance with respect to predicting accuracy, which implies that deeper neural convolutional networks as Gaussian processes do not assure a better prediction accuracy.

We could perform the same analysis for residual convolutional neural networks under the same experiment settings. The average accuracy of ResCNN-GPs is shown in Table 3.2. Still, the total number of correctly predicted points are counted and related statistics are computed. We see that the model ResCNN-GP7 has better overall accuracy on all test batches based on the training kernels of 3 randomly chosen training batches. Accuracy increases with deeper residual networks and reaches its peak at ResCNN-GP7 and drops thereafter. The same occurrence as that of CNN-GP11 happens for ResCNN-GP11, which has a relatively low prediction accuracy of 9.8%.

For both the CNN-GPs and ResCNN-GPs, they all fail when having 11 layers. It is reasonable to imagine that the failure of CNN-GP11 results in the failure of ResCNN-

Networks	Batch 1		Batch 2		Batch 3		Avg accuracy
	Num	Std	Num	Std	Num	Std	
CNN-GP3	9035	7.24	9142	**5.02**	9106	7.73	90.94%
CNN-GP5	9185	6.62	9128	7.18	9099	6.09	91.37%
CNN-GP7	**9234**	**5.12**	**9209**	5.95	9124	**4.95**	**91.89**%
CNN-GP9	9124	7.3	9118	5.76	**9222**	6.39	91.55%
CNN-GP11	980	6.36	980	8.63	980	7.25	9.8%

Table 3.1: Prediction accuracy of CNN-GPs. Num denotes the number of correctly predicted points out of 10,000 test points and Std is the standard deviation of test run based on a chosen training batch. The average accuracy is computed for each network depth over all 3 training batches.

Networks	Batch 1		Batch 2		Batch 3		Avg accuracy
	Num	Std	Num	Std	Num	Std	
ResCNN-GP3	9112	5.1	9105	5.86	9087	6.38	91.01%
ResCNN-GP5	9135	6.24	9191	5.62	9105	6.16	91.44%
ResCNN-GP7	**9238**	5.66	**9216**	6.38	9183	6.37	**92.12**%
ResCNN-GP9	9201	6.39	9134	**3.86**	**9229**	**5.51**	91.88%
ResCNN-GP11	980	**4.99**	980	5.79	980	6.86	9.8%

Table 3.2: Prediction accuracy of ResCNN-GPs. Meanings of Num and Std are as identical as those of Table 3.1.

GP11, since kernels of ResCNN-GPs just add the kernel of preceding s layers ($s = 1$ for our discussion here) to the kernel of the current layer, as shown in Equation 2.99. Both networks fail, since the exploding values for each entry in kernel matrices cause computational issues. To explain such cases, let $l = 10$ and $s = 1$, then for the kernel at the 11th layer we have

$$\begin{aligned} K^{(11)}_{\mu,res} &= K^{(9)}_{\mu,res} + \underbrace{\sigma_b^2 + \sigma_w^2 \sum_{\nu \in \mu\text{-th patch}} V^{(10)}_{\nu}}_{=K^{(11)}_{\mu,cnn}} \\ &= \underbrace{K^{(7)}_{\mu,res} + \underbrace{\sigma_b^2 + \sigma_w^2 \sum_{\nu' \in \mu\text{-th patch}} V^{(8)}_{\nu'}}_{=K^{(9)}_{\mu,cnn}}}_{=K^{(9)}_{\mu,res}} + K^{(11)}_{\mu,cnn} \\ &\cdots \\ &= K^{(1)}_{\mu,cnn} + K^{(3)}_{\mu,cnn} + K^{(5)}_{\mu,cnn} + K^{(7)}_{\mu,cnn} + K^{(9)}_{\mu,cnn} + K^{(11)}_{\mu,cnn}, \end{aligned} \tag{3.13}$$

where the hyperparameters σ_b^2, σ_w^2 and filter size for $K^{(l)}_{\mu,res}$ and $K^{(l)}_{\mu,cnn}$ should be same.

Combining $K_{\mu,cnn}^{(11)}$ and $K_{\mu,res}^{(11)}$ of all patches, respectively, at the final layer, we can see that if $K_{cnn}^{(11)}$ explodes, $K_{res}^{(11)}$ definitely explodes, leading to a final exploding scalar output. Here we examine two random images from MNIST and inspect their covariance for networks with differing number of layers. The results are shown in Table 3.3. Clearly, when the number of layers exceeds 10, the kernels of networks will explode, and the posterior predictions will be like a random guess for the label of each image, which of course is nearly 10% as the law of large numbers indicates if we make enough guesses for 10 class labels.

Covariance / Network Type / # of layers	CNN-GP	ResCNN-GP
3	9.37e+04	9.57e+04
4	6.55e+06	6.79e+06
5	4.51e+08	4.75e+08
6	3.07e+10	3.28e+10
7	2.07e+12	2.25e+12
8	1.39e+14	1.54e+14
9	9.31e+15	1.04e+16
10	6.20e+17	7.05e+17
11	NaN	NaN
12	NaN	NaN

Table 3.3: Explosion of entry values of kernel matrices. With more layers, the kernel value continuously increases until network depth reach 10 and explodes for networks whose depth is more than 10 thereafter.

We draw a conclusion from this subsection that the prediction accuracy of ResCNN-GPs is higher than that of CNN-GPs with the same layers, which indicates skip connection improve the network performance; and accuracy of the networks with more layers increase at first and then drops, and eventually both CNN-GPs and ResCNN-GPs fail due to explosion of kernels, for which the failure of ResCNN-GPs is caused by failure of CNN-GPs.

3.3 Networks with Different Number of Input Images

From now on, we will explore another factor — the number of input images, which influence the sizes of kernel matrices and concentrate on the study of how the number of input images affects the eigendecomposition and classification accuracy.

3.3.1 Eigen-analysis of Kernel Matrices

We know that for a diagonal block matrix

$$\mathbf{P} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}, \tag{3.14}$$

the eigenvalues of matrix $\mathbf{A}$ and $\mathbf{B}$ are also eigenvalues of matrix $\mathbf{P}$. However, kernel matrices $\mathbf{K}$ is generally not a form of diagonal block matrix shown as $\mathbf{P}$, thus the finding the eigen-relationship between kernel matrices and their corresponding diagonal block matrices will be very difficult in this case. We thus find the relationship through experiments.

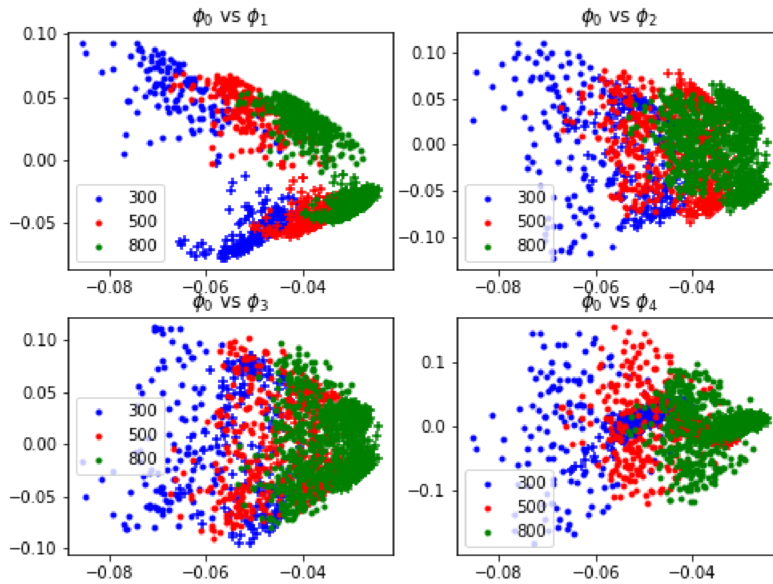
We know from Section 3.2 that the network depths have no significant impacts on the eigenvectors, and the ResCNN-GPs and CNN-GPs do not have significant differences for eigenanalysis as well. Therefore, for the simplicity of our analysis we will only focus on the CNN-GP3 with varying numbers of input images for digits 0 and 1.

We feed the CNN-GP3 with sample images of sizes 300, 500, and 800 in sequence and the pairwise plots for the top 5 eigenvectors of kernel matrices induced by CNN-GP3 are shown in Figure 3.21a. It should be noted that the latter samples contain the former samples. We see that there is a convergent trend along the eigenfunction ϕ_0 for all pairwise plots and the eigenvectors generated by digits 0 and 1 are notably separated in the first pairwise plot "ϕ_0 vs ϕ_1". Also, we inspect that with more input images the points will cluster more tightly: the green cluster generated by 800 images is tighter than the red cluster, while the red cluster is tighter than that of blue ones. Although the clusters are becoming tighter as data size grows, global geometric structures of eigenvectors are preserved for 3 datasets. It is even more intuitive when we plot the top 3 eigenvectors in 3D space, shown in Figure 3.21b. We see that the values of 3 eigenvectors are tending to converge to 0 for all 3 eigenfunctions ϕ_i for $i = 0, 1, 2$. This could be instantiated by the data from Table 3.4, in which $\mathbf{\Phi}_{300}$, $\mathbf{\Phi}_{500}$ and $\mathbf{\Phi}_{800}$ are the eigenvector matrices of kernel matrices with different input sizes, and avg and std are average and standard deviation, respectively. We see decreasing trends of absolute value of average and standard deviation for all top 3 eigenvectors when the input sizes increase, from which the smaller standard deviations explain the tighter clusters.

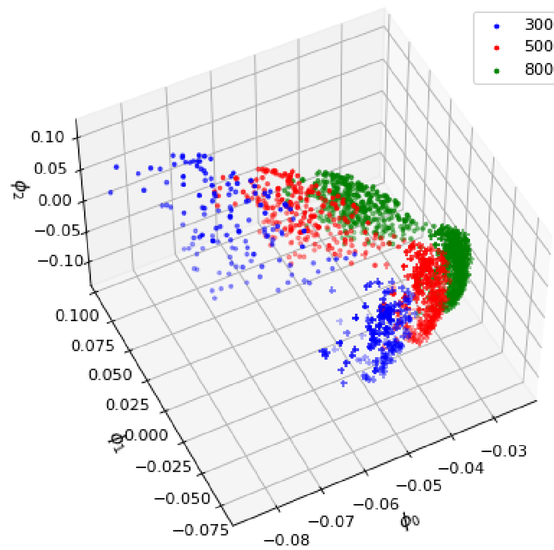
Eigenvector index		Avg	Std
1st	$\mathbf{\Phi}_{300}$[:,0]	-0.0569	0.0097
	$\mathbf{\Phi}_{500}$[:,0]	-0.0440	0.0079
	$\mathbf{\Phi}_{800}$[:,0]	-0.0348	0.0062
2nd	$\mathbf{\Phi}_{300}$[:,1]	-0.0075	0.0573
	$\mathbf{\Phi}_{500}$[:,1]	-0.0065	0.0442
	$\mathbf{\Phi}_{800}$[:,1]	-0.0052	0.0350
3rd	$\mathbf{\Phi}_{300}$[:,2]	-0.0032	0.0576
	$\mathbf{\Phi}_{500}$[:,2]	-0.0021	0.0447
	$\mathbf{\Phi}_{800}$[:,2]	-0.0014	0.0353

Table 3.4: Averages and standard deviations of top 3 eigenvectors of kernel matrices with different input sizes. For the 1st, 2nd and 3rd eigenvector, their averages all have convergent trends and standard deviations get smaller when input sizes are larger.

Figure 3.22 plots the logarithms of eigenvalues of kernel matrices induced by CNN-GP3 with different input sizes. It is known from the graph that the eigenvalues of kernel



(a) Pairwise plots of top 5 eigenvectors.



(b) Scatter plot of top 3 eigenvectors.

Figure 3.21: Eigenvectors of CNN-GP3 on digit 0 and 1 for 3 sample sizes. Digit 0 are denoted by dots and digit 1 are denotes by pluses. There are 3 samples: 300 blue points, 500 red points and 800 green points.

matrices with bigger sizes will have greater eigenvalues at the same index, which is more obvious for the largest eigenvalue of each kernel matrix, and the eigenvalues of smaller kernel matrix drop faster than those of larger kernel matrix.

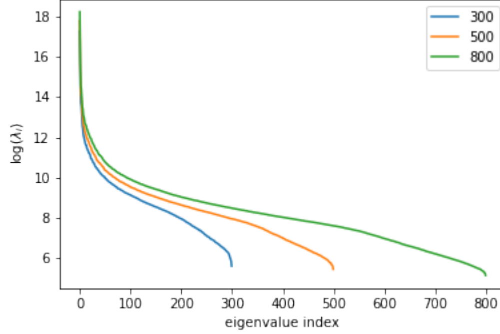


Figure 3.22: Eigenvalues of kernel matrices of CNN-GP3 with 3 different input sizes: 300, 500, and 800 images of digit 0 and 1, respectively.

With the help of eigenvalues, we can give an explanation for the convergence of eigenvectors when we feed more images to the network. Assume $\mathbf{K}_{300\times300}$ is the kernel matrix of CNN-GP3 when we feed the first 300 images consisting of digit 0 and 1, and thus we have

$$\mathbf{K}_{500\times500} = \begin{pmatrix} \mathbf{K}_{300\times300} & \mathbf{K}_{300\times200} \\ \mathbf{K}_{200\times300} & \mathbf{K}_{200\times200} \end{pmatrix} \tag{3.15}$$

when we feed 500 images consisting of digit 0 and 1, for which the first 300 images are contained and the last 200 images are newly added. When we perform eigendecompostion for both kernel matrices, we have

$$\mathbf{K}_{300\times300} = \mathbf{\Phi}_{300}\mathbf{\Lambda}_{300}\mathbf{\Phi}_{300}^{\mathrm{T}}, \tag{3.16}$$

$$\mathbf{K}_{500\times500} = \mathbf{\Phi}_{500}\mathbf{\Lambda}_{500}\mathbf{\Phi}_{500}^{\mathrm{T}}. \tag{3.17}$$

Since the first 300 eigenvalues from $\mathbf{\Lambda}_{500}$ are larger than the corresponding eigenvalues of $\mathbf{\Lambda}_{300}$ and there are extra 200 positive eigenvalues according to the Figure 3.22, the first 300 entries for every eigenvector from $\mathbf{\Phi}_{500}$ should be scaled down to fulfill block matrix $\mathbf{K}_{300\times300}$. Same holds for $\mathbf{K}_{800\times800}$.

We close this subsection by the conclusion that larger kernel matrices have more tightly clustered eigenvectors and higher eigenvalues and global geometry of datasets with different sizes are similar.

3.3.2 Performance on Classification

As we do in previous sections, we will discuss the relationship between kernel sizes and prediction accuracy. Assume both $\mathbf{X} \in \mathbb{R}^{N\times1\times28\times28}$ and $\mathbf{X}_* \in \mathbb{R}^{N^*\times1\times28\times28}$ are normalized tensors, then the size of training kernel $\mathbf{K}(\mathbf{X},\mathbf{X}) \in \mathbb{R}^{N\times N}$ is affected by

the sizes of training samples, while the sizes of testing kernel $\mathbf{K}(\mathbf{X}_*, \mathbf{X}) \in \mathbb{R}^{N^* \times N}$ is influenced by sizes of testing samples and sizes of training samples. Still, we only focus on the performance of CNN-GP3 due to the similar performance for other depths or for ResCNN-GPs. Of course, the classification accuracy is a mutual result of many factors, including network depth. For the reasons we talked about in Section 3.2, we will not consider deep networks.

Figure 3.23 shows the relationship among accuracy, training size, and testing size on classification accuracy of CNN-GP3 trained on MNIST data, which includes all ten digits from 0 to 9. To minimize any possible factors that might influence our results, we just set all the hyperparameters as the optimized parameters. It is very obvious that if the test set size is fixed we have higher accuracy once we have more training data; if the training size is fixed we have lower accuracy once we have more test data.

To put it simply, the classification accuracy is proportionate to training size and inversely proportionate to testing size. This is reasonable since more training samples will provide more information and reduce the uncertainty of our prediction and more testing samples will increase the uncertainty of our prediction due to limited information extracted from training samples.

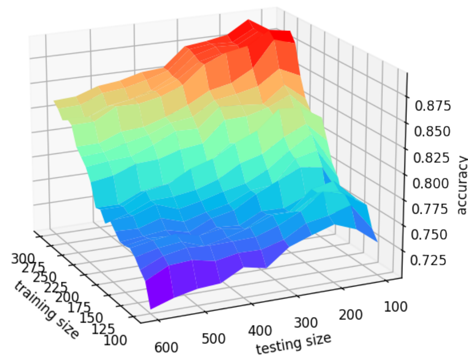


Figure 3.23: The influence of raining size and test size on classification accuracy of CNN-GP3.

4 Conclusion and Future Work

In this section, we will summarize the work we did and propose new possibilities for future work.

In Section 2, we discussed the background knowledge concerning linear operators, kernel functions, and Gaussian processes, and we then presented related work regarding the equivalence of neural networks with Gaussian processes, including Bayesian neural networks as Gaussian processes, deep neural networks as Gaussian Processes and convolutional neural networks as Gaussian processes, respectively, in the limit of infinite neurons for fully connected networks or infinite filters for convolutional neural networks.

In Section 3, we mainly explored the eigenanalysis for kernel matrices induced by CNN-GPs and investigated the performance of kernel matrices on classification accuracy. As we know, kernel matrices are affected by network architectures and their sizes are influenced by the number of input images. Of so many hyperparameters of CNN-GPs, we paid our attention to network depths, and analyzed eigenvectors and eigenvalues for (Res)CNN-GPs with differing depths, finding that network depths have only limited influence on the eigenvectors and eigenvectors are much influenced by the intrinsic similarities of image data, on the contrary, the eigenvalues are significantly influenced by network depths but the decay rate of eigenvalues is very similar for networks with different depths. Moreover, we also found that the eigenvectors and eigenvalues of ResCNN-GPs behave similarly to those of CNN-GPs, demonstrating a limited effect of skip connection in eigendecomposition of kernel matrices. Network depths affect classification accuracy as well. From experiments, we knew that networks with 7 layers have the best classification accuracy for both CNN-GPs and ResCNN-GPs and the failure of ResCNN-GPs is closely related to that of CNN-GPs. Furthermore, we explored eigenanalysis for kernel matrices induced by CNN-GPs with different numbers of input images, discovering a convergent trend for top 3 eigenvectors and higher eigenvalues at the same indices when more images are fed to the network. Through the analysis of kernel sizes and classification accuracy, we saw that networks with larger training kernel matrices and smaller testing matrices will have higher accuracy.

Thoroughly studying and understanding the properties of linear operators related to Gaussian processes induced by neural networks is interesting and challenging. Despite the study of this thesis, we still do not explore how the other hyperparameters might influence the behaviors of eigenvectors and eigenvalues. Of these hyperparameters, the nonlinearity should be much more intriguing and complicated. For future work, the study of kernel matrices resulting from changing the ReLU activation function to other choices (such as error function) would be interesting and the performance of classification on such network could be analyzed as well. In addition, future work could examine the properties of other linear operators and apply the operator for related kernels. For example, the properties of Laplacian operator ∇ applied on neural tangent kernel $\Theta(x, y; \boldsymbol{\theta})$ in equation 2.32 could be analyzed to further understand the evolutionary process of neural networks.

Bibliography

[1] Robert A. Beezer. *A First Course in Linear Algebra*. Congruent Press, 2014.

[2] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To Understand Deep Learning We Need to Understand Kernel Learning, 2018.

[3] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications (2nd Edition)*. Springer, 2005.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[7] Mikio Ludwig Braun. *Spectral Properties of the Kernel Matrix and their Ralation to Kerrnel Methods in Machine Learning*. PhD thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2005.

[8] David Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Rates of convergence for sparse variational gaussian process regression. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 862–871, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[9] David Cherney, Tom Denton, Rohit Thomas, and Andrew Waldron. *Linear Algebra*. Davis California, 2013.

[10] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21):7426–7431, 2005.

[11] J. de la Porte, B. M. Herbst, W. Hereman, and S. J. van der Walt. An Introduction to Diffusion Maps, 2008.

[12] David K. Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, University of Cambridge, 2014.

[13] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep Convolutional Networks as Shallow Gaussian Processes, 2019.

[14] Erdong Guo and David Draper. Infinitely wide tensor networks as gaussian process, 2021.

[15] Kur Hornik. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

[16] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1989.

[18] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2018.

[19] Ha Quang Mihn and Vittorio Murino. *Covariances in Computer Vision and Machine Learning.* Morgan & Claypool, 2017.

[20] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[21] Radford M. Neal. *Bayesian Learning for Neural Networks.* PhD thesis, University of Toronto, 1995.

[22] Marion Neumann, Shan Huang, Daniel E. Marthaler, and Kritian Kersting. pyGPs - A Python Library for Gaussian Process Regression and Classification. In *Journal of Machine Learning Research*, 2015.

[23] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes, 2020.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, and Vincent Dubourg. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[25] Carl Edward Rasmussen and Chritopher K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2005.

[26] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A Generalized Representer Theorem, 2001.

[27] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels.* MIT Press, 2002.

[28] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem, 1996.

[29] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Gloabl Geomertric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[30] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[31] Tim Waegemans. Adverserial Attacks on Gaussian Processes. Master's thesis, Technical University of Munich, 2021.

[32] Christopher Williams. Computing with Infinite Networks. In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[33] Chrsitopher K. I. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS 2000)*. MIT press, 2001.