Regression on Ice: Function approximation for the mathematically-inclined glaciologist.

Introduction

The rise of satellite-based remote sensing in the last couple of decades has revolutionized the way we analyze the earth and its dynamics. Some prominent examples of NASA's remote sensing programs include:

- The Gravity Recovery and Climate Experiment (GRACE), which measured variations in the Earth's gravitational field and gave new insight into the changing mass landscape of ice sheets and groundwater supply across the world.
- The Landsat mission (9 iterations, so far), which has provided high-resolution images of earth's landscape for over fifty years.
- The IceSAT mission (2 iterations, so far), which has given high-accuracy laser altimetry measurements, particularly in Greenland and Antarctica.

These programs have given us a wealth of data on the giant ice sheets of Greenland and Antarctica. In particular, they have given us a penetrating insight into the melting of the ice and the subsequent rising of sea levels.

With great volumes of data comes great responsibility. In my opinion, it requires a solid understanding of the mathematical foundations of data science. In this writeup, I would like to exposit on these foundations, in the context of modern research on ice sheet modeling. I focus on the following four topics.

- Problem Formulation and Basic Ideas in Regression
- Learning Linear Hypothesis Classes
- Gaussian Process Regression and Kriging
- On Hilbert Spaces and Kernels

Each chapter features both examples from relevant glaciology research and mathematical "curios" which invoke some higher-level themes in the theory of statistics and computation.

Contents

1 Motivation

Modern analysis of the ice sheets often relies on piecing together a series of satellite measurements to create some approximate picture of the surface and its dynamics. Consider, for instance, **the SERAC model**, which used IceSAT's six years of laser altimetry measurements to create a rasterized (i.e. pixel-based) surface model of Greenland [9]. Here's how it works:

- SERAC's spatial regression works by identifying crossover areas traced out by the satellite tracks, fitting a surface polynomial $p(x, y)$ to each of these areas, and then piecing them together for the final image. Note that these polynomials are usually degree 3, but are set to higher degree on an ad hoc basis to cover important, complicated surfaces like the Helheim and Upernavik glaciers.

- This is combined with temporal regression at the crossover points, which is assumed (mostly for simplicity) to be linear. This seems to provide space for significant improvement. We refer the reader to newer work such as the ALPS algorithm, which uses penalized splines to model complex time series of ice elevations more faithfully [11].

- They made volume change estimates, but one should note that this involved a lot more than volume integrals. They had to take into separate models for *glacial isostatic adjustment*—the earth rebounding after the weight of ice is removed—and *firn densification*—the process by which fresh snow compacts into ice. They also had to make error estimates, which they did using a *bootstrapping* technique—creating estimates based on subsets of the data, and then taking the variance of those estimates.

We make these points to illustrate that practical modeling takes a lot of flexibility and attention to detail. Data does not bend to our every assumption. Good models require both domain expertise and a mastery of the mathematical toolbox. As a mathematician-in-training, I can speak much better to the latter.

In this writeup, we abstract away the nitty-gritty issues of scattered measurements, multiple sources of error, and nontrivial ice sheet physics. We are focused on the developing not just the toolbox function approximation but an appreciation for all the mathematical insights along the way. We dive into progressively more theoretical material as the pages go by.

2 Basic Ideas in Regression

What is function approximation? Well, it's important to first decide what kinds of functions you are interested in. If the function maps into a finite set, we call this learning problem *classification*. If it maps to real numbers, we call it *regression*[1]. Since we are usually worried about real-valued inputs in glaciology (such as temperature, flow velocity, and altitude) let's formulate things in terms of regression.

2.1 Problem Formulation

Here is a condensed and somewhat informal formulation of the regression problem.

We want to estimate a real-valued function, say $f : [0,1]^d \to \mathbb{R}$ (perhaps with a function that is "simpler" in some sense). Importantly, instead of having full access to the function, we are given samples $\{(x_i, y_i)\}_{i\in[n]} \subset [0,1]^d \times \mathbb{R}$. We want to use this information find a function $\hat{f}$ which best approximates f.

There are a number of ways to express closeness of approximation. A simple way might be the mean squared error (MSE) between the predicted function and the true function, at the given sample points:

$$L(\hat{f}, f) = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}(x_i) - f(x_i))^2$$

This is a proxy for what we *really* want to know, which is the difference between the functions over not just the given points but the entire domain.

$$L(\hat{f}, f) = \int_{[0,1]^d} \Big(\hat{f}(x) - f(x)\Big)^2 dx = ||f - \hat{f}||_2^2$$

Note that $||\cdot||_p = \Big(\int (\cdot)^p \, dx\Big)^{1/p}$ denotes the p-norm (2-norm is the usual Euclidean distance, 1-norm is Manhattan distance, and so on).

There are two important and not necessarily orthogonal dimensions to this problem which I want to point out: the underlying statistical process (i.e. the sampling procedure) and the assumptions we make on our

[1] If you map into a countably infinite set like $\mathbb{N}$ (i.e. somewhere between finite and the uncountably infinite $\mathbb{R}$), you probably treat the problem more like classification, but I suppose this is a matter of taste.

2.1.1 Sampling and Bayes optimality

The process by which our samples are drawn is rather important. We often assume for the sake of simplicity that the samples are *independent and identically distributed*, like coin flips or drawing cards with replacement.

If we knew the underlying distribution which determines how our samples are drawn, we could think about loss function in terms of an "expected" squared error, where the expectation is over the randomness in the sampling process $\mathcal{D}$.

$$L(\hat{f}, f) = \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}[(\hat{f}(x_i) - y_i)^2] = \int_{[0,1]^d\times\mathbb{R}} (\hat{f}(x_i) - y_i)^2 \, d\mu_{\mathcal{D}}$$

While this loss is a good statistical lodestar, it is often impractical to calculate since $\mathcal{D}$ and its corresponding measure $d\mu_{\mathcal{D}}$ is almost always unknown.

The nice thing about this abstraction, however, is that it allows us to formulate an *optimal* regressor, sometimes referred to as the *Bayes optimal regressor.*

$$f^*(x) = \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}\Big[y_i | x_i = x\Big]$$

Without getting too much in the weeds of the highly non-trivial mathematics of conditional expectation (Cf. the Radon-Nikodym Theorem), we should take a moment to appreciate the really intuitive nature of the Bayes optimal regressor. We are trying to learn a *function* $x \mapsto y$, but we are only given a *joint distribution* between random variables X and Y. How do we turn this probabilistic information into a discrete mechanism that takes in a specific x and spits out y? The Bayes regressor says: we should first look at the marginal along $X = x$ (that's what the conditional does) and then take the expected value of Y along that marginal.

In any case: the Bayes regressor is optimal in the sense that, compared to any other regressor $\hat{f}$, it has a lower *expected* loss.

$$\mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}\Big[(f^*(x_i) - y_i)^2\Big] \leq \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}\Big[(\hat{f}(x_i) - y_i)^2\Big]$$

Note that, in practice, it is very possible that on some finite set of test points, $\hat{f}$ has a lower mean squared error of f^*. In the limit, however, due to the strong law of large numbers, the mean squared error approaches the actual expectation.

We leave the proof of the optimality of the Bayes regressor for the appendix. Note that there is a corresponding concept in classification known as the Bayes classifier.

2.1.2 Hypothesis classes: the simpler, the better

Even if you have access to the entire function—not just samples—the function approximation problem is interesting, so long as you somehow restrict the set of functions with which you approximate.

The set of functions from which we choose $\hat{f}$ is usually called the **hypothesis class**, often denoted $\mathcal{H}$. We are looking for the *best possible approximation* attainable by a given hypothesis class, which we write as the following functional optimization problem. Note that $\arg\min_{x \in X} g(x)$ means that we output the x which minimizes the function $g(x)$ (whereas $\min_{x \in X} g(x)$ would simply give us the minimum value attained by g)[2].

$$f^* = \arg\min_{\hat{f} \in \mathcal{H}} L(\hat{f}, y)$$

Intuitively, a hypothesis class is more powerful when it is small but is able to approximate many kinds of functions. Oftentimes, we use knowledge about the application domain to determine a suitable hypothesis class. For instance, if we are modeling some kind of weather pattern, we might be inclined to use some set of functions with inherently periodic behavior, such as sinusoids.

$$\mathcal{H} = \{h \text{ such that } h(x) = \sin(A_1 x_1 + b_1) + ... + \sin(A_n x_n + b_n) + c\}$$

Alternatively, we might suspect that the dependent and independent variables have a simple linear relationship, and so we work with the set of affine functions.

$$\mathcal{H} = \{h \text{ such that } h(x) = A_1 x_1 + ... + A_d x_d + b\}$$

If $d = 2$ (e.g. we are working with surfaces), we can specify the hypothesis class of functions which are degree 2 or below polynomials in the input coordinates:

$$\mathcal{H} = \{h \text{ such that } h(x) = A_1 x_1^2 + A_2 x_2^2 + A_3 x_1 x_2 + b\}$$

We note briefly that this is effectively the hypothesis class used to model patches of Greenland in the aforementioned SERAC model [9].

Notice how in the last two examples, we can describe each of the functions in the hypothesis class in terms of a few real-valued parameters, (e.g $A_1, ..., A_n, b \in \mathbb{R}$).

[2]Since we are working with real-valued functions, you might ask: what if $g(x)$ does not attain a minimum? For instance, $\min_{n \in \mathbb{N}} \frac{1}{n}$ does not exist (the infimum, meanwhile, is zero). The simple answer is: we restrict our attention to optimization problems which do not have this issue. So, technically speaking, there are some restrictions to the combinations of objective functions and domains we can pick.

Naturally, we call this *parametric* regression. You can also approach regression using *non-parametric* methods, which tend to fill in the gaps of a function based on local information rather than global structural assumptions. Two remarks on this:

- The line between parametric and non-parametric is not always clear. You might hear of splines (piecewise polynomials) as a type of *semi-parametric* regression, due to their localized nature [8]. It's sort of a matter of taste.
- Non-parametric techniques are generally more flexible and have better convergence properties than parametric techniuqes, because they make fewer assumptions about the intrinsic structure of the data. The tradeoff is that non-parametric methods require more data. There is a sort of conservation of information principle at play.

2.2 Non-parametric Regression

The classic non-parametric regression technique is k**-nearest neighbors**. The idea is simple: given a test point $x_{\text{test}} \in [0,1]^d$, we assign y_{test} to be the average of the k nearest training points. If we are measuring closeness using the Euclidean norm, then, in a mix of math notation and pseudocode, we may write the following.

$$\hat{f}(x_{\text{test}}) = y_{\text{test}} = \frac{1}{k}\ \text{sum}\Big(\text{k arg}\min_{x \in X_{\text{train}}} ||x_{\text{test}} - x||_2\Big)$$

Rather than taking a hard cutoff of nearest neighbors, we could use a softer, weighted scheme. In this scheme, every training point makes an impact, but closer points make more of an impact than points which lie further away. For instance, we might model locality in terms of a Gaussian distribution, where the parameter h denotes the variance.

$$\hat{f}(x_{\text{test}}) = \frac{\sum_{x \in X_{\text{train}}} y_i \cdot \exp(-||x_{\text{test}} - x||^2/2h^2)}{\sum_{x \in X_{\text{train}}} \exp(-||x_{\text{test}} - x||^2/2h^2)}$$

This is an example of so-called **kernel regression**, where the kernel refers to the function that mediates our weighted average. The above example uses the Gaussian kernel, but we can easily abstract this to some function K with a locality-controlling parameter h:

$$\hat{f}(x_{\text{test}}) = \frac{\sum_{x \in X_{\text{train}}} y_i \cdot K_h(||x_{\text{test}} - x||)}{\sum_{x \in X_{\text{train}}} K_h(||x_{\text{test}} - x||)}$$

Kernel regressors have nice convergence properties. For instance, there are results which suggest that, as the window size shrinks ($h \to 0$) and the number of random samples goes to infinity ($n \to \infty$), but at a faster rate than the shrinkage of the window ($hn \to \infty$; if the window shrunk faster, then there might be no points in the window to make predictions), then the predictor $\hat{f}$ converges to the Bayes optimal regressor. This is known as a **consistency result**: it establishes that our model does the right thing as we have more data points to work with.

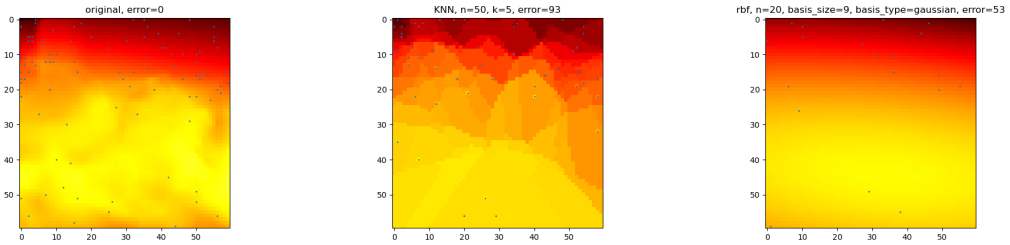

Figure 1: Regression of sea surface temperatures. Left is the original image, middle is k-nearest neighbors, right is kernel regression with Gaussian kernel.

2.3 On the word *kernel*

There are a few fundamentally different and important concepts that the word *kernel* expresses across mathematics.

1. (Algebra) The kernel of a linear map $f : V \to W$ is the set of elements in V which are mapped to the zero element in W. Symbolically, $\ker(f) = \{v \in V : f(v) = 0\}$. The same idea is at play when you describe the kernel of a group homomorphism or a morphism of categories.

2. (Functional Analysis) In a Hilbert space $\mathcal{H}$ (i.e. a vector space of functions with some sense of an inner product), a kernel $k : \mathcal{H} \times \mathcal{H} \mapsto \mathbb{R}$ is simply a symmetric, positive-semidefinite map. Famously, every kernel is "native" to a (often infinite-dimensional) reproducing kernel Hilbert space. See chapter 4 for more details.

3. (Statistics) In the loosest use of the word, a kernel is just a non-negative "window function" which is zero or near-zero outside a small neighborhood. A kernel is sort of like a "measuring device" you can apply to a function to get local information. You usually apply it using an operation called *convolution*.

The last definition is usually the one lurking in the shadows of machine learning terminology such as the aforementioned kernel regression and the so-called "kernel trick." But we will see later how important the function analysis type kernel is. And if you have ever taken linear algebra, you will appreciate the importance of kernels.

2.4 Curio: Voronoi, Delaunay, and Duality

Suppose $d = 2$ (i.e. our inputs are points on a grid) and $k = 1$ (we are only assigning things to the nearest neighbor). Then the nearest neighbor assignments create a tesselation of the plane, where each tile is the neighborhood of a given training point.

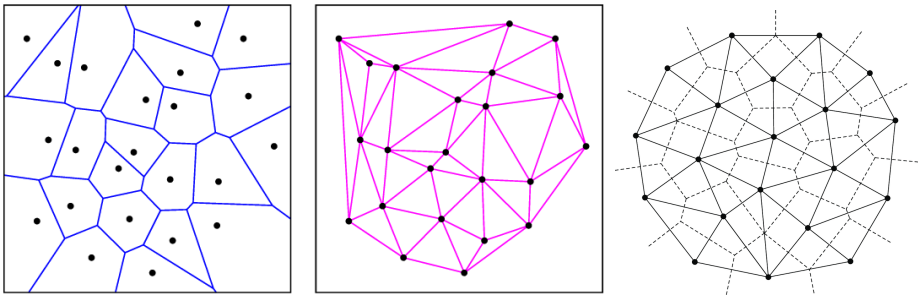

Figure 2: The Voronoi diagram (left) is the tesselation created by 1-nearest neighbors. If we view it as a graph then, the Delaunay triangulation (middle) is its dual. On the right, we show the two superimposed.

If we are using Euclidean distance, then each tile is a convex polytope (does not cave in on itself and has flat edges). If you view the straight-line edges of these tiles as edges of a graph, then (unsurprisingly) the Voronoi diagram is a graph.

Now forget about nearest neighbors for a moment and consider a different problem over our set of test points on the plane: say we want to connect the dots in such a way to form a *triangulation* (i.e. each face of the graph is a triangle). Say we want this triangulation to maximize the smallest angle in this triangulation, i.e. avoid sliver-like triangles. The winner of this optimization problem is the so-called **Delaunay triangulation**.

It turns out that the Delaunay triangulation is the **dual graph** of the Voronoi diagram, in the sense that the faces of the Voronoi graph are the nodes of the Delaunay graph, and corresponding edges cross each other (at ninety degrees in the precise geometric construction). See the illustration above.

It is somewhat remarkable that these different optimizations have this sort of symmetry between them. The term mathematicians use for this kind of behavior is called *duality*. It is an exciting theme that shows up all across mathematics. We note the following examples.

- The dual V^* of a vector space V, which is the set of linear maps from V to its underlying base field.
- The dual of a constrained optimization problem, which, under certain conditions (e.g. Slater's Conditions for convex optimization) is equivalent to the original, primal formulation.
- The duality of open and closed sets in topology; trace and determinant of a linear operator; the time and frequency domains in Fourier analysis; intermediate field extensions and subgroups of the corresponding Galois group; kernel and image of group homomorphisms; and so and so forth.

Keep your eyes peeled for duality. It's quite beautiful.

3 Linear Hypothesis Classes

In this chapter, we discuss the standard front-line approach to function approximation: linear regression. After recalling some elementary linear algebra, we explain the standard solution to linear regression. Then we discuss the simple tweaks one makes to upgrade this method in order to run polynomial, spline, and radial basis function regression. Finally, we discuss error estimation techniques and hyperparameter tuning for the more advanced techniques.

3.1 Lines and Linearity

Intuitively, we all know what a line is. The first descriptor that may come to mind is: straight. Lines are straight. They also seem to extend in both directions, as opposed to their one-track counterparts, the rays, or their finite cousins, the line segment.

If you are a bit more descriptive, you might say that a line $f : \mathbb{R} \mapsto \mathbb{R}$ is a function that can be written $y = mx + b$. It turns out, however, that the better word for this kind of function is *affine*. If we are being precise, we reserve the word linear for functions which cross the origin. In other words, we have:

$$\text{(affine) } f(x) = mx + b \qquad \text{(linear) } f(x) = mx$$

If we are generalizing this to higher-dimensional functions $f : \mathbb{R}^d \mapsto \mathbb{R}$, then we have, for $A \in \mathbb{R}^{1\times d}$ and $b \in \mathbb{R}$:

$$\text{(affine) } f(x) = Ax + b \qquad \text{(linear) } f(x) = Ax$$

If the function is $f : \mathbb{R}^d \mapsto \mathbb{R}^n$, then it would be the same except $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$. The pattern should be clear. And for most scientific applications, this conception of the line is sufficient. But from a mathematical perspective, it is more of a description than a real definition.

Here is a more mathematically palatable definition of a line.

Definition: A function $f : \mathbb{R} \mapsto \mathbb{R}$ is a *linear map* (or *line*) if it satisfies:

1. (homogeneity) $f(ca) = cf(a)$ for all $a, c \in \mathbb{R}$.
2. (additivity) $f(a) + f(b) = f(a + b)$ for all $a, b \in \mathbb{R}$

Together, we call these two properties *linearity*.

You might think this is more than enough to qualify such a simple concept. But, if you are familiar with linear algebra, you will know that there is a much richer story at play, where the main characters are *vector spaces.* I will provide two descriptions of vector spaces: one in lay terms and the other in the form of an algebraic definition.

- A vector space V is a set of abstract items (lists or tables of numbers, functions, et cetera). I call these items vectors. I don't know much about them, but I do know this: if I add two vectors, it's still a vector: $v_1 + v_2 \in V$. If I multiply a vector by a real number, it's still a vector: $cv \in V$. There is a zero vector, such that $\vec{0} + v = v$ for all $v \in V$.
- More precisely, a vector space V over a field (of scalars) F has two operations: vector addition $+ : V \times V \mapsto V$ and scalar multiplication $+ : F \times V \mapsto V$. V is an abelian (i.e. commutative) group under vector addition, and scalar multiplication is a left ring action on the group of vectors. (If F were a ring instead of a field, then we call this set a module rather than a vector space).

3.1.1 Vector Spaces of Functions

I make this point of discussing vector spaces because many interesting classes of functions that we want to use for regression form vector spaces over the field of real numbers. Here are some useful examples $\mathbb{R} \mapsto \mathbb{R}$.

- **Polynomials** are linear combinations of monomials. Polynomials of degree d are parameterized by a vector of coefficients $(a_1, ..., a_d)$.

$$f(x) = \sum_{i=0}^{d} a_i x^i$$

  The space of polynomials of arbitrarily large degree form an infinite-dimensional vector space. If we restrict the dimension, it becomes finite-dimensional.

- **Sinusoids** are linear combinations of shifted and scaled sine functions (or cosines: same difference). We can parameterize them in terms of a set of amplitudes, frequencies, and shifts $\{(a_i, c_i, t_i)\}_{i \in [n]}$.

$$f(x) = \sum_{i=0}^{n} a_i \sin\Big(c_i(x - t_i)\Big)$$

- **Splines** are piecewise polynomials, with "break-points" parameterized by a so-called *knot vector* $(t_1, ...t_m)$. We can write them somewhat perversely as follows, where $\mathbf{1}_{[a,b]}(x)$ is the indicator function of $x \in [a, b]$.

$$f(x) = \sum_{i=0}^{n} \sum_{j=0}^{m} a_{ij} x^i \mathbf{1}_{[t_i, t_{i+1})}(x)$$

The nicer way of writing out a basis for splines is to use the so-called Cox-de Boor recursive formula.

- **Radial basis functions** (RBFs) are linear combinations of some radially symmetric function $\phi(x)$. They are parameterized by function centers $(t_i)_{i\in[n]}$, e.g.

$$f(x) = \sum_{i=0}^{n} a_i \phi_i(x) \qquad \phi_i(x) = \lambda \exp(-||x - t_i||^2)$$

3.1.2 Universal Approximation

The fact that these are vector spaces make them amenable to least squares linear regression. But there is a deeper reason that these function spaces are good candidates for function reconstruction, and it has to do with a concept known as *universal approximation.*

Essentially, any continuous function $f : [a, b] \mapsto \mathbb{R}$ can be approximated, with arbitrarily good precision, by these kinds of functions. The classical result, to this end, is the Weierstrass approximation theorem, stated below.

Theorem 1 *(Weierstrass, 1885) If $f : [a, b] \mapsto \mathbb{R}$ is continuous, then for all $\epsilon > 0$, there exists a polynomial $p(x)$ such that:*

$$||f - p||_\infty = \sup_{x\in[a,b]} |f(x) - p(x)| < \epsilon$$

The Stone-Weierstass theorem famously allows us to generalize this to other spaces of functions, including sinusoids.

However, even if a space of functions has universal approximability, it might not be all that good at function approximation in practice. This is because, to meet a given precision level, you may need to use very high degree polynomials or very high-frequency sinusoids. However, in practice, when fitting functions, we cannot work with the entire function space. We specify a a maximum degree for our polynomials and a maxinum frequency of our sinusoids. We bound the number of knots in our splines and the number of centers we use for our radial basis function. If we didn't, then our code would never terminate.

With that humbling reminder, let us descend from the cloud of the continuous world, back into a discussion of more practical, discrete computation.

3.2 Ordinary Least Squares

If your hypothesis class consists of affine functions, then finding the best least-squares fit from inside this hypothesis class is famously easy. Recall that we are learning a function $[0,1]^d \mapsto \mathbb{R}$. It is affine in its input $x = (x_1, ..., x_d) \in [0,1]^d$ if:

$$h_\omega(x) = \sum_{j=1}^{d} x_j \omega_j$$

We can compute how well it does on the training set $\{(x_i, y_i)\}_{i\in[n]} \subset [0,1]^d \times \mathbb{R}$, by adding up the squared error accumulated by the test function on each (x_i, y_i) pair.

$$L(h_\omega, y) = \sum_{i=1}^{n} \Big(h_\omega(x_i) - y_i\Big)^2 = \sum_{i=1}^{n} \Big(\sum_{j=1}^{d} x_{ij}\omega_j - y_i\Big)^2$$

We can express this in matrix notation as follows, where $X \in \mathbb{R}^{n\times d}$ encodes all of the inputs as row vectors, and $y \in \mathbb{R}^n$ has the dependent variable.

$$L(h_\omega, y) = ||X\omega - y||^2$$

We want to minimize this function with respect to $\omega \in \mathbb{R}^d$, the parameterization of h. We call this the **least-squares problem**.

$$\arg\min_{\omega\in\mathbb{R}^d} ||X\omega - y||^2$$

Without much thinking, we could look for stationary points $\frac{\partial L}{\partial \omega} = 0$ and hope for the best. The outcome of this computation is as follows.

$$\frac{\partial}{\partial\omega}||X\omega - y||^2 = X^T(X\omega - y) = 0$$

$$\implies \boxed{\hat{\omega} = (X^TX)^{-1}X^Ty}$$

It turns out that, if X^TX is indeed invertible[3], then this is the square-loss-minimizing ω. The mathematical argument for this requires a bit of work: you need to show that L_ω is convex in ω. Then we apply a well-known property of convex functions: namely, that all local minima are global minima.

[3]If it is not invertible, then you can use the Moore-Penrose pseudo-inverse.

3.2.1 On Convexity and Optimization

Convexity (like linearity) is an example of remarkably well-behaved mathematics. When optimization problems exhibit convexity, they make our lives a lot easier. Let us introduce convexity from the ground up.

Recall that a *linear combination* of vectors $x_1, ..., x_n$ is of the following form, where $\lambda_1, ..., \lambda_n \in \mathbb{R}$ are the coefficients.

$$\lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_n x_n$$

Linear combinations of one vector form a line. Linear combinations of two (independent) vectors form a plane. And so on.

There are special types of linear combinations which form more restrictive spaces.

- A *conic combination* is a linear combination such that $\lambda_1, ..., \lambda_n$ are non-negative. As the name suggests, these combinations form a space that looks like a cone.
- A *convex combination* is a linear combination such that the $\lambda_1, ..., \lambda_n$ are non-negative and they add up to one, i.e. $\sum_{i=1}^{n} \lambda_i = 1$. The space created by the convex combination of $x_1, ..., x_n$ is called a *convex hull*.

Now we introduce two concepts that play with the idea of convex combinations.

Definition 1 *$f : \mathbb{R} \mapsto \mathbb{R}$ is convex if the following holds:*

$$\forall x, y \in \mathbb{R} \ , \ \lambda \in (0, 1) \qquad f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

Convexity is a very natural concept if you look at it from the right lens. It tells you that f "plays nicely" with averaging, in the sense that there is a consistent relationship between the average of two function values and the function evaluated at the average value.

The image you should have in mind is a parabola, but be aware that convex functions come in a number of shapes and sizes. Importantly, affine functions are convex (and in their case, the convexity inequality is tight).

Definition 2 *A convex set $S \subset \mathbb{R}$ is a set which is closed under convex combinations. Visually speaking, if you draw a line connecting any two points in the set, the line lies within the set*[4]*. Symbolically:*

$$\forall x, y \in S \ , \ \lambda \in (0, 1) \qquad \lambda x + (1 - \lambda)y \in S$$

[4]So we could have alternatively said that a convex function is such that the space above the function $\{(x, y) : y \geq f(x)\}$ is a convex set.

Each of these concepts are important to convex optimization. Without getting into the weeds of optimization theory, I would like to point out that there are two ingredients to any optimization problem:

- *The objective function* which we are trying to minimize or maximize.
- *The feasible set.* This is the set from which we choose our parameters.

In the case of linear regression, the objective is $f(\omega) = ||X\omega - y||^2$, and the feasible set is the whole space $\mathbb{R}^d$. This is known as an *unconstrained* optimization problem. If we constrained the feasible set to, say $[0,1]^d$, then this is a *constrained* optimization problem.

3.2.2 Maximum-Likelihood Formulation

A tell-tale sign of good mathematics is what I would call *robustness under change in perspective*: there are many ways of looking at the same thing. The least squares problem is a shining example of this.

You may have seen the generative (or Bayesian, if you will) formulation of least squares, which proceeds roughly as follows.

Assume that your points $\{(x_i, y_i)\}_{i \in [n]}$ are noise-perturbed samples of an actual linear function. Importantly, we assume that the noise terms are all independent, identically distributed Gaussians.

$$\forall i \qquad y_i = \omega \cdot x_i + \epsilon_i \qquad \epsilon_i \sim \mathcal{N}(0,1)$$

Then, as the argument goes, it makes sense (computationally and intuitively) to maximize the following quantity, often denoted as the *likelihood*.

$$\arg\max_{\omega} \ \mathbb{P}(y_1, ..., y_n \mid \omega)$$

This is a conditional probability: we read the objective function as "the probability of the observations $y_1, ..., y_n$, given some setting of the parameters ω."

The claim is the following: given our assumptions about the noise

$$\arg\max_{\omega} \ \mathbb{P}(y_1, ..., y_n \mid \omega) = \arg\min_{\omega} ||X\omega - y||^2$$

We leave the proof for the appendix. Some interesting questions you might ask, after reading the proof, are: what if we change the noise structure? Is there still an equivalence here?

3.2.3 Gauss-Markov Theorem

The claim above is, in some sense, a special case of a more general theorem about linear regression. It turns out that the correspondence between least squares and likelihood maximization can occur even if the noise structure is not just normally distributed.

Theorem 2 *If $y = X\omega + \epsilon$, with the noise $\epsilon = (\epsilon_1, ..., \epsilon_d)$ a random vector such that:*

- *(mean zero) $\mathbb{E}(\epsilon_i) = 0$ for all i.*
- *(homoscedastic) $Var(\epsilon_i^2) = \mathbb{E}(\epsilon_i^2) = \sigma^2 < \infty$ for all i.*
- *(uncorrelated) $Cov(\epsilon_i, \epsilon_j) = 0$ for all i, j.*

Then the ordinary least squares estimator $\hat{\omega} = (X^T X)^{-1} X^T y$ has the lowest sampling variance within the class of linear unbiased estimators. In other words, it minimizes $\mathbb{E}||X\omega - y||^2$, where the expectation is over the random noise.

3.3 Generalizations of Linear Regression

There are a number of ways to adapt the technique of linear regression. First of all, we can fit all kinds of functions to data, rather than just straight lines. Secondly, we can apply penalties to enforce certain kinds of structure on our estimate.

3.3.1 Linear Hypothesis Classes

We can use least squares regression to fit not just lines but polynomials, splines, and sinusoids to all kinds of data. How we do this requires a clever transformation of the input matrix X. We build up the intuition for this carefully.

Recall that, if we are learning a function $\mathbb{R} \mapsto \mathbb{R}$, then linear regression looks particularly simple. We are finding the coefficients ω which allow us to express y as a linear combination of sampled points $x_1, ..., x_m \in \mathbb{R}$.

$$X\omega - y = \begin{pmatrix} x_1 & \dots & x_m \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

If we are learning a function $\mathbb{R}^d \mapsto \mathbb{R}$ (this is sometimes called ordinary *multilinear regression*), then we are learning to express y as a linear combination of sampled *vectors* $x_1, ..., x_m \in \mathbb{R}^d$.

$$X\omega - y = \begin{pmatrix} | & & | \\ x_1 & \dots & x_m \\ | & & | \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Learning a to express a function $\mathbb{R} \mapsto \mathbb{R}$ in terms of, say, polynomials up to degree m, looks very similar to multi-linear regression. Say we have three test points $x_0, x_1, x_2 \in \mathbb{R}$. Since our basis functions are effectively $\{1, x, x^2, ..., x^m\}$, each column of the matrix is an application of the basis function to the vector (x_0, x_1, x_2).

$$X\omega - y = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

More generally, we can fix any finite basis $\{\phi_1(x), ..., \phi_m(x)\}$ and run least squares on the following quantity.

$$\begin{pmatrix} | & & | \\ \phi_1(x) & \dots & \phi_m(x) \\ | & & | \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Whether those ϕ's are splines, polynomials, sinusoids, or some mix theoreof, is effectively a design choice. The basis functions you choose also can be multi-dimensional. Once should just note that the size of the feature matrix X grows faster when you do fancy basis functions in high dimensions.

3.3.2 Regularization

As we mentioned in the first chapter on regression, it is important that our approximations are as simple as possible. This is not only intuitive but backed by certain theoretical results (Cf. Ockham's Razor theorem, in computational learning theory).

The process of enforcing simplicity in our regression estimates is often referred to as **regularization**. In the context of linear regression, there are two canonical types of regularization, referred to as ridge and lasso regression.

We can characterize these types of regression by the penalties that we add to the function. In *ridge regression*, we penalize based on the squared Euclidean norm of the coefficient vector, $||\omega||_2^2 = \sum_{i=1}^n \omega_i^2$.

$$\textbf{Ridge (L2)} \qquad \arg\min_{\omega} ||X\omega - y||^2 + \lambda||\omega||_2^2$$

In so-called l*asso regression*, we penalize based on the L1 norm, $||\omega||_1 = \sum_{i=1}^{n} |\omega_i|$.

$$\textbf{Lasso (L1)} \qquad \arg\min_{\omega} ||X\omega - y||^2 + \lambda||\omega||_1$$

Despite their somewhat similar formulations, these two penalties enforce very different behavior on the resulting solution. The major difference is that lasso regression enforces *sparsity* in the output vector, in the sense that the lasso-optimal solution will have mostly zero entries. In practice, you choose to use lasso when you expect only a few of the input variables to play a role. A prime example

The trade-off is that lasso regression, despite being a convex optimization problem, has no closed-form solution and usually needs to be approximated. Ridge regression, meanwhile, is easy to calculate and has the following closed-form solution:

$$\omega_{\text{r}} = (X^T X + \lambda I)^{-1} X^T y$$

The nice thing about this solution is that $X^T X + \lambda I$ is always invertible for $\lambda > 0$, so we do not need to worry about matrix singularity issues the same way we do for normal OLS.

We note briefly that the design of good penalties is a bit of an art and there is much more variety out there than the two canonical regularizers mentioned above. This is particularly true when implementing broader function classes such as polynomials or splines.

Consider the rather fancy difference-based penalty used in the Approximation by Localized Penalized Splines (ALPS) [11], a popular regression technique for glaciology. This was adapted from a technique by statisticians Eilers and Marx [3] proposed almost twenty years earlier. Let B be the spline basis functions, packaged discretely as a matrix.

$$\arg\min_{\omega} \Big(||B\omega - y||^2 + \lambda P \Big) \qquad P = ||\Delta^q \omega||^2$$

The idea is to penalize q-order changes in the output function. The intutive way of doing this would involve looking at the q-th order derivative of the output function $X\omega$ and minimizing its integral. However, this numerical integral is computationally costly. [3] found that looking at the q-th order difference of successive coefficients in the ω vector was a sufficient proxy for this behavior.

Here is an explicit example: for $q = 1$, the vector $\Delta^q(\omega)$ can be written as follows.

$$(\Delta^1 \omega)_j = \omega_j - \omega_{j+1} \qquad j \in \{0, ..., n-1\}$$

For $q = 2$, we have:

$$(\Delta^2 \omega)_j = \omega_j - 2\omega_{j+1} + \omega_{j+2} \qquad j \in \{0, ..., n-2\}$$

For $q = 3$, we have:

$$(\Delta^3\omega)_j = \omega_j - 3\omega_{j+1} + 3\omega_{j+2} - \omega_{j+3} \qquad j \in \{0, ..., n-3\}$$

See a pattern? (Hint: Pascal's Triangle). We leave it as an exercise to the reader consider the more general formulas for q-order difference and write the Δ^q operator out explicitly as a matrix.

3.4 Curio 1: Greenland Volume Balance Error Estimation

3.5 Curio 2: Free Knot Linear Splines and Neural Networks

As we have seen, spline regression with fixed knots is easily solved using least-squares regression. However, this masks a highly non-trivial aspect of spline fitting. If you try to optimize the *placement of the knots*, you find yourself mired in a rather complicated, non-convex optimization procedure, with many potentially poor local minima. This is captured by David Jupp calls the "lethargy theorem" [6].

A cultural aside: splines are not as popular as they used to be in the statistics and machine learning literature. Nowadays, *neural networks* are the rage. Neural networks are class of functions that roughly mimic the way neurons communicate with each other. They are demanding in terms of computation, but if you give them the time and data, they can do amazing things: from beating the best human chess player, to modeling the complex process of protein folding.

I would like to let you in on a shockingly underappreciated fact (or perhaps well-kept secret): in their usual implementation, **neural networks are splines**. That's right: the most popular machine learning method in recent memory is an old friend in disguise.

To be more specific, neural networks with the so-called ReLU (rectified linear unit; $f(x) = \max\{0, x\}$) activation is a free-knot linear spline [2]. In other words, it is a piecewise linear function. The interesting thing about it is that it can have a lot of pieces, and it seems to manage the placement of those pieces rather well.

To get a better idea of this, let's define exactly what we are talking about.

Definition 3 *A ReLU neural network $N(W, L) : \mathbb{R}^d \mapsto \mathbb{R}^{d'}$ of width W, length L, input dimension d, and output dimension d' is defined via the following recursion, where x_0 is the input vector and x_{L+1} is the output vector.*

$$x_k = \max\left\{ W_k x_{k-1} + b_{k-1},\ 0 \right\} \qquad 1 \leq k \leq L+1$$

In other words, a neural network is simply a composition of linear functions, except between these compositions we "chop off" the nonzero elements. It is precisely this chopping off which turns the linear into the piecewise linear.

In order to have any luck at analyzing, we need language to think about not just individual functions but families of functions. Hence the following definitions.

Definition 4 *$Y^{W,L}(d, d')$ denotes the set of neural networks with width W, length L, input dimension d, and output dimension d'.*

Definition 5 *$\Sigma(D)$ denote the set of free knot linear splines $\mathbb{R} \mapsto \mathbb{R}$ with at most D breakpoints (i.e. D discontinuities in the derivative).*

Daubechies and friends [1] showed that for single hidden layer ReLU nets ($L = 1$), we have the following *strict* inclusions between the aforementioned function classes.

$$\Sigma(W - 1) \subset Y^{W,1}(1, 1) \subset \Sigma(W)$$

Of course, $L = 1$ is a pretty restrictive condition. What happens when we look at truly deep neural networks? This is where the magic kicks in. It turns out that the number of breakpoints grows *exponentially* in L, giving us the following inclusion.

$$Y^{W,L}(1, 1) \subset \Sigma((W + 1)^L)$$

It turns out that this upper bound is tight, in the sense that you can come up with an example of a neural net which has this many break points. Telgarsky showed this using so-called sawtooth functions [12]. However, there are many free knot linear splines with this many break points which do not conform to a neural network representation. It remains an open question how we might characterize the ones which can be expressed as neural networks.

Now, what happens with higher dimensions, i.e. $d, d' > 1$? Splines are a bit harder to describe in higher dimensions, so we instead call the larger class of functions which neural networks occupy to be *continuous piecewise linear functions*, sometimes abbreviated CPwL. We have the following elementary result describing this relationship.

Theorem 3 *[2] Every $f \in Y^{W,L}(d, d')$ is a continuous piecewise linear function subordinate to a polytope partition with at most 3^{WL} cells.*

The proof uses the idea of activation patterns. The polytope partition is derived from zones where the $m = WL$ hidden layer neurons are either positive, negative, or

zero. For $\nu \in \{-1, 0, 1\}^m$ and pre-activations of each neuron $z(x) = (z_k(x), k \in [n])$, we construct the set:

$$\Sigma_\nu = \{x \in \mathbb{R}^d : \text{sgn}(z_k(x)) = \nu\}$$

Since there are at most 3^n possible ν with Σ_ν non-zero measure, this is our upper bound on the size of the partition. We can trivially reduce this to 2^n and have almost-everywhere subordination to a polytope partition (since the contribution of the zero activations is measure zero).

In any case, the view of neural networks as piecewise linear functions has proved useful in theoretical analyses. But big questions still remain. Personally, I want to know how neural networks avoid the optimization issues faced by free knot linear splines. It seems to have something to do with what some researchers call the *implicit regularization* abilities of neural networks. How is the neural network limiting its use the exponentially many breakpoints, in order to efficiently approximate functions?

4 Gaussian Processes

One of the biggest problems in regression is finding out how best to express the error of our estimates. It turns out, if we model our predictions as a family of inter-related random variables, we tend to have a much easier time quantifying uncertainty. This is the basic motivation of Gaussian process regression.

4.1 Random Variables and the Gaussian

Without getting too much into the measure-theoretic weeds, let us recall some fundamental ideas in probability theory.

A *random variable* is a function. In particular, it is a measurable function (generalization of a continuous function, in some sense) associating subsets of some set Σ (the "state space") to their probabilities in $\mathbb{R}$. A simple example might be a Rademacher random variable.

$$f : \mathcal{P}(\{-1, 1\}) \mapsto \mathbb{R}$$

Where the state space $\Sigma = \mathcal{P}(\{-1, 1\})$, the power set of $\{-1, 1\}$ (i.e. the set of all subsets). This random variable assigns probabilities as follows.

$$f(\phi) = 0 \qquad f(\{-1\}) = f(\{1\}) = 1/2 \qquad f(\{-1, 1\}) = 1$$

The *distribution* of a random variable, roughly speaking, describes the underlying probabilistic structure of a random variable. The Rademacher random variable a *discrete* distribution, due to its finiteness. The exponential distribution, on the other hand, is referred to as a *continuous* distribution.

The Gaussian, also known as the normal distribution or bell curve, is a very special continuous distribution. Thus, we use integration (rather than summation) in order to describe how it measures sets. For instance, in order to find the probability that a Gaussian random variables lies in the set $[-3, 3]$, we write

$$\int_{-3}^{3} p_{\mu,\sigma}(x)dx = \int_{-3}^{3} \Big[\frac{1}{\sqrt{2\pi\sigma^2}} \exp(-(x-\mu)^2/2\sigma^2)\Big]dx$$

where $p_{\mu,\sigma}(x)$ is the is Gaussian's *probability density function*[5]. μ is the mean. σ^2 is the variance. The Gaussian is special for a number of reasons. It is the continuous distribution of maximal entropy for unbounded variables, the fixed point of the

[5]We note briefly that not all continuous functions have probability density functions. By the Radon-Nikodym theorem, a random variable f has a density function if and only if its cumulative distribution function $F(x) = \mu(f^{-1}((-\infty,)))$ is absolutely continuous

Fourier transform, and perhaps most importantly, the limiting distribution in the Central Limit Theorem.

4.2 The Multivariate Gaussian

The natural next step in our discussion is to introduce the multivariate Gaussian random variable, whose state space is $\mathbb{R}^d$ instead of just $\mathbb{R}$. The probability density function looks very similar to the one-dimensional counterpart.

$$p_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^d} \det(\Sigma)^{1/2} \exp((x-\mu)^T \Sigma (x-\mu))$$

The vector $\mu \in \mathbb{R}^d$ is the mean and $\Sigma \in \mathbb{R}^{d \times d}$ is the covariance matrix. In other words, each element of Σ stores the covariance between individual coordinates, i.e.

$$\Sigma_{ij} = \mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)]$$

The diagonal entries describe the variance of each coordinate itself.

$$\Sigma_{ii} = \mathbb{E}[(x_i - \mu_i)^2] = \mathbb{E}[x_i^2] - \mathbb{E}[x_i]^2 = \mathrm{Var}(x_i)$$

A multi-variate Gaussian random variable $x = (x_1, ..., x_n) \sim \mathcal{N}(\mu, \Sigma)$, has remarkable closure properties under *marginalization* and *conditioning*, i.e. when you apply these operations you still get a multivariate normal (albeit with difference mean and covariance) [4].

- **Marginalization** of a random vector involves isolating some subset of the indices, and getting rid of the rest of them by integration. For instance, suppose we are looking at X, Y disjoint subsets of the coordinates, such that $X \cup Y = \{x_1, ..., x_n\}$. Suppose $p_{X,Y}$ is the probability density for the entire multivariate Gaussian. Then the marginal over X is given by "integrating out" the Y part.

$$p_X(x) = \int_Y p_{X,Y}(x, y)\ dy$$

- **Conditioning** involves asking: what does one set of the indices look like, given what the others look like? The formula for the density function of $X|Y$, given below, should be intuitive if you have ever seen conditional probability before.

$$p_{X|Y}(x, y) = \frac{p_{X,Y}(x, y)}{p_Y(y)}$$

The important thing to know is that $X|Y$ and X are each random variables in their own right.

With a strong understanding of random variables, probability densities, and the multivariate Gaussian distribution, the concept of a Gaussian process is almost a bit too effortless to define.

Definition 6 *A Gaussian process is a (potentially infinite) set of random variables* $\{X_\alpha\}_{\alpha \in \mathcal{A}}$ *such that any finite subset has a multivariate Gaussian distribution.*

Note that we are not specifying that this family of random variables is independent or identically distributed. Indeed, this is where the fun of Gaussian processes really comes into play. The way we define the covariance structure over space and time determines everything.

In the next section, we look at a prominent example of Gaussian process regression known as Kriging. It was developed in some sense before the theory of Gaussian processes was fully developed, so not all of the information we offered above will be necessary. But it offers nice context nonetheless.

4.3 Kriging

Kriging interpolation was developed in the context of a search for gold. Since then it has become the standard in geostatistical surface estimation.

The mathematical formulation of it all should not distract from the simple initial goal: similar to kernel regression, as described in Chapter 1, we want to express the surface as a linear combination of the given measurements. For instance, if $(z_i)_{i \in [n]}$ are the measured points and $x \in \mathbb{R}^d$ is the coordinate, then:

$$Z(x) = \sum_{i=1}^{n} \alpha_i(x) z_i$$

Importantly, we treat the output of Z as a Gaussian random variable, and $\{Z(x)\}_{x \in [0,1]^d}$ as a family of inter-related Gaussian random variables. As such, it is a stochastic process, more specifically a Gaussian process.

In the same vein as seeking out the smallest possible hypothesis class, we make two assumptions in order to restrict the kinds of functions that $\alpha_i(x)$ can be. Let $\mu \in \mathbb{R}$ and $\gamma : \mathbb{R}^+ \mapsto \mathbb{R}^+$ be a monotone, continuous function we call the variogram.

- (O1) In expectation, the surface is flat: $\mathbb{E}(Z(x)) = \mu$ for all x.

- (O2) The covariance structure is radially symmetric.

$$\text{Var}\Big[Z(x+h) - Z(x)\Big] = \gamma(||h||)$$

Applying the definition of variance and the linearity of expectation, we may equivalently write:

$$\begin{aligned}\text{Var}\Big[Z(x+h) - Z(x)\Big] &= \mathbb{E}\Big[\Big(Z(x+h) - Z(x)\Big)^2\Big] - \Big[\mathbb{E}(Z(x+h) - Z(x))\Big]^2 \\ &= \mathbb{E}\Big[Z(x+h)^2 + Z(x)^2 - 2Z(x+h)Z(x)\Big] \\ &= 2\gamma(0) - 2\,\mathbb{E}\Big(Z(x+h)Z(x)\Big)\end{aligned}$$

Hence, we have the following explicit formula for the covariance between two points in the random field:

$$\mathbb{E}\Big(Z(x+h)Z(x)\Big) = 2\gamma(0) - \gamma(||h||)$$

If we know what the variogram is, then there is a reasonable condition we can enforce on the α_i: namely, to minimize the variance in the estimate Z. So the first step for us, naturally, is to calculate that variance. For any point x_0 on the surface, the variance is given as follows.

$$\sigma^2_{s_0} = \mathbb{E}\Big[\Big(Z^*(x_0) - Z(x_0)\Big)^2\Big]$$

In the next step we proceed to express our prediction as a linear combination of the measured points.

$$\mathbb{E}\Big[\Big(\sum_i \alpha_i Z(x_i) - Z(x_0)\Big)^2\Big]$$

Then we multiply things out and apply linearity of expectation to simplift things.

$$\mathbb{E}\Big[\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j(Z(x_i) - Z(x_j))^2 - \sum_{i=1}^n \alpha_i(Z(x_i) - Z(x_0))^2\Big]$$

$$\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j\mathbb{E}[(Z(x_i) - Z(x_j)]^2 - \sum_{i=1}^n \alpha_i\mathbb{E}[Z(x_i) - Z(x_0)]^2\Big]$$

If we let $v_{ij} = \gamma(||x_i - x_j||)$, which is a matrix we effectively can compute beforehand, we have the following:

$$\sigma_{s_0}^2 = \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j v_{ij} - \sum_i \alpha_i v_i$$

We are looking for α which minimizes the variance, subject to the constraint that the α_i's add up to one. We can write this compactly as follows:

$$\min_{\alpha} \sigma_{s0}^2 \qquad \text{s.t.} \qquad \sum_{i=1}^{n} \alpha_i = 1$$

Applying the first fundamental insight of Lagrange optimization, we notice that the above problem can be rewritten as minimum over a maximum. In the literature this is called the **primal problem**, hence the denotation p^*.

$$p^* = \min_{\alpha} \max_{\lambda} \Big(\sigma_{s0}^2 + \lambda\Big(1 - \sum_{i=1}^{n} \alpha_i\Big)\Big)$$

The second step is to ask the question: what happens if I switch the min and max? We call this the **dual problem**.

$$d^* = \max_{\lambda} \min_{\alpha} \Big(\sigma_{s0}^2 + 2\lambda\Big(1 - \sum_{i=1}^{n} \alpha_i\Big)\Big)$$

In general, we always have $d^* \leq p^*$. This is called weak duality. In specific situations, we have the more helpful connection that $d^* = p^*$ identically, a condition known as strong duality. This is the case, for instance, for convex objective functions with affine constraints (these are known as Slater's conditions for convex optimization). We happen to fit these conditions for the above problem.

From here, it makes sense to differentiate with respect the variables at play, set those derivatives to zero, and then solve the system. We have enough solutions such that we can determine the optimal α uniquely.

The function at hand is the following.

$$L(\alpha, \lambda) = \sigma_{s0}^2 + 2\lambda\Big(\sum_{i=1}^{n} \alpha_i - 1\Big)$$

Setting the derivatives with respect to each α_i and λ gives us the following linear system of equations, which can be solved using classic matrix inversion.

$$\sum_{j=1}^{n} \alpha_j \gamma(||h_{ij}||) + 2\lambda\alpha_i = \alpha_i \gamma(||h_{i0}||) \quad \text{for each } 1 \leq i \leq n$$

$$\sum_i \alpha_i = 1$$

For a suitably chosen variogram γ, the above should, in principle, yield a unique solution. I say in principle because, in practice, it is often the case that the linear system is undetermined, i.e. the matrix we are trying to invert is not full rank. Usually this is due to a lack of data, or sampled data that is too close together.

These are relevant issues for satellite data, especially something like IceSAT, where the measurement tracks are relatively tight and end up missing somewhat sizable swaths of area.

4.4 Curio: Wide Neural Networks

Wide neural networks have deep connections with Gaussian processes [7]. The exact mathematics can get a little hairy but the intuition behind it is actually quite simple, if you have a good grasp of the famous central limit theorem.

Theorem 4 (Central Limit Theorem) *Consider $\{X_i\}_{i\in\mathbb{N}}$ a sequence of random variables which are independent, identically distributed, and have bounded variance $\mathbb{E}(X_i^2) = \sigma^2 < 0$. Let $\mu = \mathbb{E}(X_i)$. Then the following converges in distribution to a standard Gaussian random variable.*

$$\lim_{n\to\infty} \frac{\sum_{i=1}^n (X_i - \mu)}{\sigma\sqrt{n}} \sim \mathcal{N}(0,1)$$

There is a related result for random *vectors* as opposed to random variables, which tells us that a sequence of random variables with the same properties states above converges to a multivariate Gaussian distribution.

The idea is that, in the infinite width limit and with the right initialization, the activation layers of a neural network are Gaussian processes. This may seem surprising until you realize that the usual way we initialize a neural network (i.e. the way we set the weights before we start training) is by setting each weight and bias in the network independently and according to the same distribution, with $1/\sqrt{N}$ normalization. So the central limit theorem applies directly. The activation for the k-th layer is given by:

$$x_k = \sum_{i=1}^{\infty} W_{ik}x_{k-1} + b_k = \lim_{n\to\infty} \Big(\frac{\tilde{b}_k + \sum_{i=1}^{\infty} \tilde{W}_{ik}x_{k-1}}{\sqrt{n}}\Big)$$

It may seem a bit contrived, but this observation (which has existed for decades, by the way) is actually pretty powerful and has led to a recent breakthrough discovery

in the theoretical study of neural networks. This discovery is known as the *neural tangent kernel* [5]. It allows to use the tools of functional analysis and Hilbert spaces to better explain the effectiveness of gradient descent in training neural networks. We describe some of this further in the next chapter.

5 Function Spaces

This chapter is intended as an introduction to some of the fundamental ideas and intuitions of functional analysis. Although some of the topics here are quite abstract, this is meant to be accessible and insightful to the *practitioner*. Mathematics is supposed to help us better organize our thoughts. A deeper understanding of what functions are and how they act gives us better ability to build new, exciting models. The curio at the end of this chapter is an example.

5.1 Functions as vectors

One of the fundamental conceptual leaps in higher mathematics is the idea that *functions are vectors*. You can imagine this in two different ways.

- (Lists) In practical settings, we often think of vectors as lists of numbers, say $v = (v_1, ..., v_n) \in \mathbb{R}^n$. In some sense, this means that I am associating each of the numbers 1 through n with a distinct real number. So there is some function $\tilde{v} : \{1, ..., n\} \mapsto \mathbb{R}$ corresponding to v. This makes the step towards generalization a little easier: why limit ourselves to discrete domains? Why not $\tilde{v} : \mathbb{N} \mapsto \mathbb{R}$ (i.e. sequences) or $\tilde{v} : \mathbb{R} \mapsto \mathbb{R}$ (functions)?

- (Algebra) Say we are looking at the set of functions $\mathbb{R} \mapsto \mathbb{R}$, which we indeed may write as $\mathbb{R}^{\mathbb{R}}$. This is a vector space because its elements satisfy the classic linearity property, under point-wise addition.

  $$(cf + bg)(x) = c \cdot f(x) + b \cdot g(x) \qquad \forall f, g \in \mathbb{R}^{\mathbb{R}}, b, c \in \mathbb{R}$$

  The more algebraic way of saying this is that, if f, g are functions, then linear combinations $cf + bg$ are also functions. It seems almost too obvious to record, but it is really quite profound.

Now, the space of all functions on the real line $\mathbb{R}^{\mathbb{R}}$ is unimaginably huge and incredibly ugly. Almost none of these functions are continuous, and you can find all kinds of examples of functions which are, say, continuous everywhere but differentiable nowhere (in other words, very, very rough).

Exercise: Find a function $\mathbb{R} \mapsto \mathbb{R}$ which is discontinuous everywhere except at one point. (**Hint**: Leverage the density of the rationals in the reals.)

Badly behaved functions are harder to approximate. It is easier when they have intrinsic structure like continuity or differentiability. Consider *Taylor's approximation theorem* which tells us how to approximate a function on a certain neighborhood, in terms of linear combinations of its derivatives. In general, the more derivatives we can take, the better our approximation can become.

5.2 Topology, and why it matters

In order to appreciate Hilbert spaces, which are a special class of functions, we must delve a bit into point-set topology. Topology is the fundamental notion that gives us concepts such as neighborhoods, closeness, and limits. When we talk about *spaces*, we really mean *topological spaces* on the most fundamental level. Things like Hilbert and Banach spaces have additional structure on top of the basic topology.

We offer a laundry list of definitions, partly for reference, and partly because it is useful to see how the concepts build upon each other in this very calculated way.

- A **topological space** is a pair $(X, \mathcal{O})$ with $\mathcal{O} \subset \mathcal{P}(X)$ the set of so-called open sets of X. These open sets satisfy three properties:

  (1) $\phi \in \mathcal{O}$.

  (2) $\mathcal{O}$ is closed under finite intersections.

  (3) $\mathcal{O}$ is closed under arbitrary (finite or infinite) unions.

  It might seem surprising that a topology is all you need to define the concept of a limit or a continuous function. The topology is arguably one of the most triumphant definitions in all of mathematics.

- A **metric space** is a pair (X, d) with X an arbitrary set and $d : X \times X \mapsto \mathbb{R}$ a distance function satisfying three properties:

  (positive definiteness) $d(x, y) \geq 0$ and is only exactly zero if $x = y$.

  (symmetry) $d(x, y) = d(y, x)$.

  (triangle inequality) $d(x, y) \leq d(x, z) + d(z, y)$.

  Note that a metric space is automatically a topological space. We say the metric *induces a topology.*

- A **normed vector space** is a pair $(X, ||\cdot||)$, with X a vector space (say over some field K) and $||\cdot|| : X \mapsto \mathbb{R}$ a norm, meaning it obeys these three properties:

  (positive definiteness) $||x|| \geq 0$ and $||x|| = 0 \iff x = \vec{0}$.

  (homogeneity) $||c \cdot x|| = |c| \cdot ||x||$ for $c \in K$, $x \in X$.

  (triangle inequality) $||x + y|| \leq ||x|| + ||y||$.

  Note that a norm induces a metric as follows: $d(x, y) = ||x - y||$.

- An **inner product space** is a pair $(X, \langle \cdot, \cdot \rangle)$ with X a vector space over either the real numbers $\mathbb{R}$ or the complex numbers $\mathbb{C}$ and $\langle \cdot, \cdot \rangle : X \times X \mapsto \mathbb{R}$ an inner product, meaning it satisfies the following:

  (positive definiteness) $\langle x, x \rangle > 0$, $\langle x, x \rangle = 0 \iff x = 0$.

  (conjugate symmetry) $\langle x, y \rangle = \overline{\langle y, x \rangle}$ with $\overline{a + bi} = a - bi$.

  (linearity in the first coordinate) $\langle \lambda a + b, y \rangle = \lambda \langle a, y \rangle + \langle b, y \rangle$

  Naturally, an inner product induces a norm $||x|| = \sqrt{\langle x, x \rangle}$

Introductory texts tend to gloss over this hierarchy of spaces. But this hierarchy is sort of at the heart of functional analysis. Without it, it is hard to appreciate the more complicated constructions, such as Hilbert spaces, which we now introduce.

5.3 Hilbert Spaces

A Hilbert space $\mathcal{H}$ is a *complete* inner product space. Completeness is a purely topological notion: it effectively tells us that sequences of numbers that *should* converge, do converge. In some sense, it tells us that our space does not have holes (for instance, the rational numbers $\mathbb{Q}$ are not complete, but the real numbers are).

Some canonical examples of Hilbert spaces include:

- $\mathbb{C}^n$ with standard inner product:

$$\langle x, y \rangle = \sum_{i=1}^{n} \overline{x_i}\, y_i$$

- Continuous complex-valued functions on $[a, b]$, denoted $C[a, b]$, with:

$$\langle f, g \rangle = \int_{[a,b]} \overline{f(x)} g(x) dx$$

- Complex-valued matrices $\mathbb{C}^{n \times m}$ with inner product given via trace.

$$\langle A, B \rangle = \text{tr}(A^* B) = \text{tr}(\overline{A^T} B) = \sum_{k=1}^{n} \overline{A}_{ki} B_{ki}$$

$$L_p = \{f : ||f||_p < \infty\}$$

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

Theorem (Equivalence of Kernel and Embedding): $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a map $\Phi : \mathcal{X} \mapsto \mathcal{H}$ such that

Reproducing Property: for all $x \in X$ there exists a uniwue $K_x \in H$ with the reproducing property:

$$f(x) = L_x(f) = \langle f m K_x \rangle_H$$

Example: multi-scale basis functions and applications in digital elevation modeling. (Patra Shekhar)

5.4 Curio (Multi-scale RKHS)

Multiscale Models are known to be successful in uncovering and analyzing the structures in data at different resolutions. In the current work we propose a feature driven Reproducing Kernel Hilbert space (RKHS), for which the associated kernel has a weighted multiscale structure. For generating approximations in this space, we provide a practical forward-backward algorithm that is shown to greedily construct a set of basis functions having a multiscale structure, while also creating sparse representations from the given data set, making representations and predictions very efficient. We provide a detailed analysis of the algorithm including recommendations for selecting algorithmic hyper-parameters and estimating probabilistic rates of convergence at individual scales. Then we extend this analysis to multiscale setting, studying the effects of finite scale truncation and quality of solution in the inherent RKHS. In the last section, we analyze the performance of the approach on a variety of simulation and real data sets, thereby justifying the efficiency claims in terms of model quality and data reduction.

[10]

6 Appendix

6.1 MLE Formulation of Least Squares

$$\begin{aligned}
\arg\max_{\omega} \ \mathbb{P}(y_1, ..., y_n \mid \omega) &= \arg\max_{\omega} \ \prod_{i=1}^{n} \mathbb{P}(y_i \mid \omega) \\
&= \arg\max_{\omega} \ \log\Big(\prod_{i=1}^{n} \mathbb{P}(y_i \mid \omega)\Big) \\
&= \arg\max_{\omega} \ \sum_{i=1}^{n} \log\Big(\mathbb{P}(y_i \mid \omega)\Big)
\end{aligned}$$

References

[1] Ingrid Daubechies et al. "Nonlinear approximation and (deep) ReLU networks". In: *Constructive Approximation* 55.1 (2022), pp. 127–172.

[2] Ronald DeVore, Boris Hanin, and Guergana Petrova. "Neural network approximation". In: *Acta Numerica* 30 (2021), pp. 327–444.

[3] Paul HC Eilers and Brian D Marx. "Flexible smoothing with B-splines and penalties". In: *Statistical science* 11.2 (1996), pp. 89–121.

[4] Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. "A visual exploration of gaussian processes". In: *Distill* 4.4 (2019), e17.

[5] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: Convergence and generalization in neural networks". In: *Advances in neural information processing systems* 31 (2018).

[6] David LB Jupp. "Approximation to data by splines with free knots". In: *SIAM Journal on Numerical Analysis* 15.2 (1978), pp. 328–343.

[7] Jaehoon Lee et al. "Deep neural networks as gaussian processes". In: *arXiv preprint arXiv:1711.00165* (2017).

[8] David Ruppert, Matt P Wand, and Raymond J Carroll. *Semiparametric regression*. 12. Cambridge university press, 2003.

[9] Tony Schenk and Beata Csatho. "A new methodology for detecting ice sheet surface elevation changes from laser altimetry data". In: *IEEE Transactions on Geoscience and Remote Sensing* 50.9 (2012), pp. 3302–3316.

[10] Prashant Shekhar and Abani Patra. "A forward–backward greedy approach for sparse multiscale learning". In: *Computer Methods in Applied Mechanics and Engineering* 400 (2022), p. 115420.

[11] Prashant Shekhar et al. "Alps: a unified framework for modeling time series of land ice changes". In: *IEEE Transactions on Geoscience and Remote Sensing* 59.8 (2020), pp. 6466–6481.

[12] Matus Telgarsky. "Representation benefits of deep feedforward networks". In: *arXiv preprint arXiv:1509.08101* (2015).