4

Representations and features

The starting point for prediction is the existence of a vector x from which we can predict the value of y. In machine learning, each component of this vector is called a *feature*. We would like to find a set of features that are good for prediction. Where do features come from in the first place?

In much of machine learning, the feature vector x is considered to be given. However, features are not handed down from first principles. They had to be constructed somehow, often based on models that incorporate assumptions, design choices, and human judgments. The construction of features often follows human intuition and domain specific expertise. Nonetheless, there are several principles and recurring practices we will highlight in this chapter.

Feature representations must balance many demands. First, at a population level, they must admit decision boundaries with low error rates. Second, we must be able to optimize the empirical risk efficiently given the current set of features. Third, the choice of features also influences the generalization ability of the resulting model.

There are a few core patterns in feature engineering that are used to meet this set of requirements. First, there is the process of turning a measurement into a vector on a computer, which is accomplished by *quantization and embedding*. Second, in an attempt to focus on the most discriminative directions, the binned vectors are sorted relative to their similarity to a set of likely patterns through a process of *template matching*. Third, as a way to introduce robustness to noise or reduce and standardize the dimension of data, feature vectors are compressed into a low, fixed dimension via *histograms and counts*. Finally, *nonlinear liftings* are used to enable predictors to approximate complex, nonlinear decision boundaries. These processes are often iterated, and often times the feature generation process itself is tuned on the collected data.

Measurement

Before we go into specific techniques and tricks of trade, it's important to recognize the problem we're dealing with in full generality. Broadly

speaking, the first step in any machine learning process is to numerically represent objects in the real world and their relationships in a way that can be processed by computers.

There is an entire scientific discipline, called measurement theory, devoted to this subject. The field of measurement theory distinguishes between a measurement procedure and the target *construct* that we wish to measure.[1,2,3] Physical temperature, for example, is a construct and a thermometer is a measurement device. Mathematical ability is another example of a construct; a math exam can be thought of as a procedure for measuring this construct. While we take reliable measurement of certain physical quantities for granted today, other measurement problems remain difficult.

It is helpful to frame feature creation as measurement. All data stems from some measurement process that embeds and operationalizes numerous important choices.[4] Measurement theory has developed a range of techniques over the decades. In fact, many measurement procedures themselves involve statistical models and approximations that blur the line between what is a feature and what is a model. Practitioners of machine learning should consult with experts on measurement within specific domains before creating ad-hoc measurement procedures. More often than not there is much relevant scholarship on how to measure various constructs of interest. When in doubt it's best to choose constructs with an established theory.

Human subjects

The complexities of measurement are particularly apparent when our features are about human subjects. Machine learning problems relating to human subjects inevitably involve features that aim to quantify a person's traits, inclinations, abilities, and qualities. Researchers often try to get at these constructs by designing surveys, tests, or questionnaires. However, much data about humans is collected in an ad-hoc manner, lacking clear measurement principles. This is especially true in a machine learning context.

Featurization of human subjects can have serious consequences. Recall the example of using prediction in the context of the criminal justice system. The COMPAS recidivism risk score is trained on survey questions designed using psychometric models to capture archetypes of people that might indicate future criminal behavior. The exam asks people to express their degree of agreement with statements such as "I always practice what I preach," "I have played sick to get out of something," and "I've been seen by others as cold and unfeeling."[5] Though COMPAS features have been used to predict recidivism, they have been shown to be no more predictive than using age, gender, and past criminal activity.[6]

Machine learning and data creation involving human subjects should be ethically evaluated in the same manner as any other scientific investigation with humans. Depending on context, different ethical guidelines and regulations exist that aim at protecting human research subjects. The 1979 Belmont Report is one ethical framework, commonly applied in the United States. It rests on the three principles of respect for persons, beneficence, and justice. Individuals should be treated as autonomous agents. Harm should be minimized, while benefits should be maximized. Inclusion and exclusion should not unduly burden specific individuals, as well as marginalized and vulnerable groups.

Universities typically require obtaining institutional approval and detailed training before conducting human subject research. These rules apply also when data is collected from and about humans online.

We advise any reader to familiarize themselves with all applicable rules and regulations regarding human subject research at their institution.

Quantization

Signals in the real world are often continuous and we have to choose how to discretize them for use in a machine learning pipeline. Broadly speaking, such practices fall under the rubric of *quantization*. In many cases, our goal is to quantize signals so that we can reconstruct them almost perfectly. This is the case of high resolution photography, high fidelity analog-to-digital conversion of audio, or perfect sequencing of proteins. In other cases, we may want to only record skeletons of signals that are useful for particular tasks. This is the case for almost all quantizations of human beings. While we do not aim to do a thorough coverage of this subject, we note quantization is an essential preprocessing step in any machine learning pipeline. Improved quantization schemes may very well translate into improved machine learning applications. Let us briefly explore a few canonical examples and issues of quantization in contemporary data science.

Images

Consider the raw bitmap formatting of an image. A bitmap file is an array indexed by three coordinates. Mathematically, this corresponds to a *tensor* of order 3. The first two coordinates index space and the third indexes a color channel. That is, x_{ijk} denotes the intensity at row i, column j, and color channel k. This representation summarizes an image by dividing two dimensional space into a regular grid, and then counting the quantity of each of three primary colors at each grid location.

This pixel representation suffices to render an image on a computer screen. However, it might not be useful for prediction. Images of the same scene with different lighting or photographic processes might end up being quite dissimilar in a pixel representation. Even small translations of two images might be far apart from each other in a pixel representation. Moreover, from the vantage point of linear classification, we could train a linear predictor on any ordering of the pixels, but scrambling the pixels in an image certainly makes it unrecognizable. We will describe transformations that address such shortcomings of pixel representations in the sequel.

Text

Consider a corpus of n documents. These documents will typically have varying length and vocabulary. To embed a document as a vector, we can create a giant vector for every word in the document where each component of the vector corresponds to one dictionary word. The dimension of the vector is therefore the size of the dictionary. For each word that occurs in the document we set the corresponding coordinate of the vector to 1. All other coordinates corresponding to words not in the document we set to 0.

This is called a *one-hot encoding* of the words in the dictionary. The one-hot encoding might seem both wasteful due to the high dimension and lossy since we don't encode the order of the words in the document. Nonetheless, it turns out to be useful in a number of applications. Since typically the language has more dictionary words than the length of the document, the encoding maps a document to a very sparse vector. A corpus of documents would map to a set of sparse vectors.

Template matching

While quantization can often suffice for prediction problems, we highlighted above how this might be too fine a representation to encode when data points are similar or dissimilar. Often times there are higher level patterns that might be more representative for discriminative tasks. A popular way to extract these patterns is *template matching*, where we extract the correlation of a feature vector x with a known pattern v, called *template*.

At a high level, a template match creates a feature x' from the feature x by binning the correlation with a template. A simple example would be to fix a template v and compute

$$x' = \max\left\{v^T x, 0\right\}.$$

We now describe some more specific examples that are ubiquitous in pattern classification.

Fourier, cosine, and wavelet transforms

One of the foundational patterns that we match to spatial or temporal data is sinusoids. Consider a vector in $\mathbb{R}^d$ and the transformation with k-th component

$$x'_k = |v_k^T x| \,.$$

Here the ℓ-th component of v_k is given by $v_{k\ell} = \exp(2\pi i k\ell/d)$. In this case we are computing the magnitude of the *Fourier transform* of the feature vector. This transformation measures the amount of oscillation in a vector. The magnitude of the Fourier transform has the following powerful property: Suppose z is a translated version of x so that

$$z_k = x_{(k+s) \bmod d}$$

for some shift s. Then one can check that for any v_k,

$$|v_k^T x| = |v_k^T z| \,.$$

That is, the magnitude of the Fourier transform is *translation invariant*. There are a variety of other transforms that fall into this category of capturing the transformation invariant content of signals including cosine and wavelet transforms.

Convolution

For spatial or temporal data, we often consider two data points to be similar if we can translate one to align with another. For example, small translations of digits are certainly the same digit. Similarly, two sound utterances delayed by a few milliseconds are the same for most applications. *Convolutions* are small templates that are translated over a feature figure to count the number of occurrences of some pattern. The output of a convolution will have the same spatial extent as the input, but will be a "heat map" denoting the amount of correlation with the template at each location in the vector. Multiple convolutions can be concatenated to add discriminative power. For example, if one wants to design a system to detect animals in images, one might design a template for heads, legs, and tails, and then linear combinations of these appearances might indicate the existence of an animal.

Summarization and histograms

Histograms summarize statistics about counts in data. These serve as a method for both reducing the dimensionality of an input and removing noise in the data. For instance, if a feature vector was the temperature in a location over a year, this could be converted into a histogram of temperatures which might better discriminate between locations. As another example, we could downsample an image by making a histogram of the amount of certain colors in local regions of the image.

Bag of words

We could summarize a piece of text by summing up the one-hot encoding of each word that occurs in the text. The resulting vector would have entries where each component is the number of times the associated word appears in the document. This is a *bag of words* representation of the document. A related representation that might take the structure of text better into account might have a bag of words for every paragraph or some shorter-length contiguous context.

Bag of words representations are surprisingly powerful. For example, documents about sports tend to have a different vocabulary than documents about fashion, and hence are far away from each other in such an embedding. Since the number of unique words in any given document is much less than all possible words, bag-of-words representations can be reasonably compact and sparse. The representations of text as large-dimensional sparse vectors can be deployed for predicting topics and sentiment.

Downsampling/pooling

Another way to summarize and reduce dimension is to *locally* average a signal or image. This is called downsampling. For example, we could break an image into 2x2 grids, and take the average or maximum value in each grid. This would reduce the image size by a factor of 4, and would summarize local variability in the image.

Nonlinear predictors

Once we have a feature vector x that we feel adequately compresses and summarizes our data, the next step is building a prediction function $f(x)$. The simplest such predictors are linear functions of x, and linear functions are quite powerful: all of the transformations we have thus far discussed

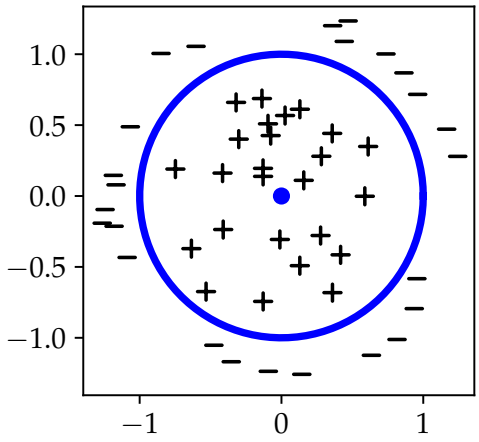

Figure 1: A cartoon classification problem for polynomial classification. Here, the blue dot denotes the center of the displayed circle.

in this chapter often suffice to arrange data such that linear decision rules have high accuracy.

However, we oftentimes desire further expressivity brought by more complex decision rules. We now discuss many techniques that can be used to build such nonlinear rules. Our emphasis will highlight how most of these operations can be understood as embedding data in spaces where linear separation is possible. That is: we seek a nonlinear transformation of the feature vector so that linear prediction works well on the transformed features.

Polynomials

Polynomials are simple and natural nonlinear predictors. Consider the dataset in the figure below. Here the data clearly can't be separated by a linear function, but a quadratic function would suffice. Indeed, we'd just use the rule that if $(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq c_3$ then (x_1, x_2) would have label $y = 1$. This rule is a quadratic function of (x_1, x_2).

To fit quadratic functions, we only need to fit the *coefficients* of the function. Every quadratic function can be written as a sum of quadratic monomials. This means that we can write quadratic function estimation as fitting a *linear* function to the feature vector:

$$\Phi_2^{\text{poly}}(x) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 \end{bmatrix}^T$$

Any quadratic function can be written as $w^T \Phi_2^{\text{poly}}(x)$ for some w. The

map Φ_2^{poly} is a *lifting function* that transforms a set of features into a more expressive set of features.

The features associated with the quadratic polynomial lifting function have an intuitive interpretation as crossproducts of existing features. The resulting prediction function is a linear combination of pairwise products of features. Hence, these features capture co-occurrence and correlation of a set of features.

The number of coefficients of a generic quadratic function in d dimensions is

$$\binom{d+2}{2},$$

which grows quadratically with dimension. For general degree p polynomials, we could construct a lifting function $\Phi_p^{\text{poly}}(x)$ by listing all of the monomials with degree at most p. Again, any polynomial of degree p can be written as $w^T \Phi_p^{\text{poly}}(x)$ for some w. In this case, $\Phi_p^{\text{poly}}(x)$ would have

$$\binom{d+p}{p}$$

terms, growing roughly as d^p. It shouldn't be too surprising to see that as the degree of the polynomial grows, increasingly complex behavior can be approximated.

How many features do you need?

Our discussion of polynomials led with the motivation of creating nonlinear decision boundaries. But we saw that we could also view polynomial boundaries as taking an existing feature set and performing a nonlinear transformation to embed that set in a higher dimensional space where we could then search for a linear decision boundary. This is why we refer to nonlinear feature maps as *lifts*.

Given expressive enough functions, we can always find a lift such that a particular dataset can be mapped to any desired set of labels. How high of a dimension is necessary? To gain insights into this question, let us stack all of the data points $x_1, \ldots, x_n \in \mathbb{R}^d$ into a matrix X with n rows and d columns. The predictions across the entire dataset can now be written as

$$\widehat{y} = Xw\,.$$

If the x_i are linearly independent, then as long as $d \geq n$, we can make *any* vector of predictions y by finding a corresponding vector w. For the sake of *expressivity*, the goal in feature design will be to find lifts into high

dimensional space such that our data matrix X has linearly independent columns. This is one reason why machine learning practitioners lean towards models with more parameters than data points. Models with more parameters than data points are called *overparameterized*.

As we saw in the analysis of the perceptron, the key quantities that governed the number of mistakes in the perceptron algorithm were the maximum norm of x_k and the norm of the optimal w. Importantly, the dimension of the data played no role. Designing features where w has controlled norm is a domain specific challenge, but has nothing to do with dimension. As we will see in the remainder of this book, high dimensional models have many advantages and few disadvantages in the context of prediction problems.

Basis functions

Polynomials are an example of *basis functions*. More generally, we can write prediction functions as linear combinations of B general nonlinear functions $\{b_k\}$:

$$f(x) = \sum_{k=1}^{B} w_k b_k(x)$$

In this case, there is again a lifting function $\Phi_{\text{basis}}(x)$ into B dimensions where the kth component is equal to $b_k(x)$ and $f(x) = w^T \Phi_{\text{basis}}(x)$. There are a variety of basis functions used in numerical analysis including trigonometric polynomials, spherical harmonics, and splines. The basis function most suitable for a given task is often dictated by prior knowledge in the particular application domain.

A particularly useful set in pattern classification are the *radial basis functions*. A radial basis function has the form

$$b_z(x) = \phi(\|x - z\|)$$

where $z \in \mathbb{R}^d$ and $\phi : \mathbb{R} \to \mathbb{R}$. Most commonly,

$$\phi(t) = e^{-\gamma t^2}$$

for some $\gamma > 0$. In this case, given $z_1, \ldots, z_k$, our functions take the form

$$f_k(x) = \sum_{j=1}^{k} w_j e^{-\gamma \|x - z_j\|^2}.$$

Around each anchor point z_k, we place a small Gaussian bump. Combining these bumps with different weights allows us to construct arbitrary functions.

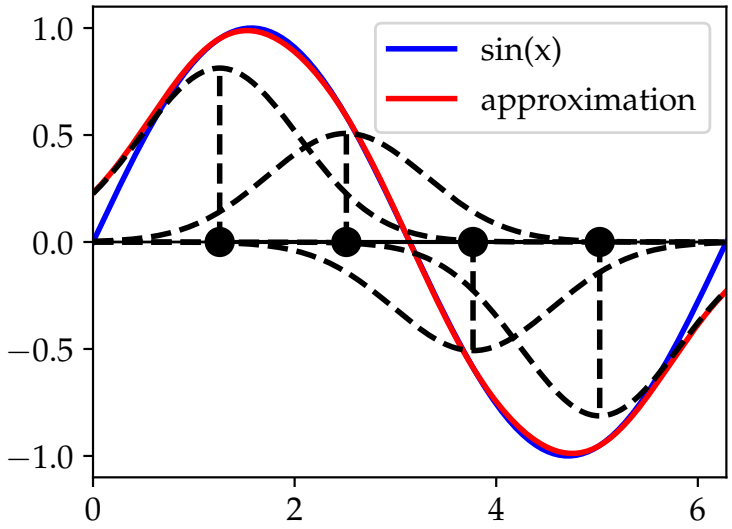

Figure 2: Radial Basis Function approximation of sin(x). We plot the four Gaussian bumps that sum to approximate the function.

How to choose the z_k? In low dimensions, z_k could be placed on a regular grid. But the number of bases would then need to scale exponentially with dimension. In higher dimensions, there are several other possibilities. One is to use the set of training data. This is well motivated by the theory of *reproducing kernels*. Another option would be to pick the z_k at random, inducing *random features*. A third idea would be to search for the best z_i. This motivates our study of *neural networks*. As we will now see, all three of these methods are powerful enough to approximate any desired function, and they are intimately related to each other.

Kernels

One way around high dimensionality is to constrain the space of prediction function to lie in a low dimensional subspace. Which subspace would be useful? In the case of linear functions, there is a natural choice: the span of the training data. By the fundamental theorem of linear algebra, any vector in $\mathbb{R}^d$ can be written as sum of a vector in the span of the training data and a vector orthogonal to all of the training data. That is,

$$w = \sum_{i=1}^{n} \alpha_i x_i + v$$

where v is orthogonal to the x_i. But if v is orthogonal to every training data point, then in terms of prediction,

$$w^T x_i = \sum_{j=1}^{n} \alpha_j x_j^T x_i \,.$$

That is, the v has no bearing on in-sample prediction whatsoever. Also note that prediction is only a function of the dot products between the training data points. In this section, we consider a family of prediction functions built with such observations in mind: we will look at functions that expressly let us compute dot products between liftings of points, noting that our predictions will be linear combinations of such lifted dot products.

Let $\Phi(x)$ denote any lifting function. Then

$$k(x, z) := \Phi(x)^T \Phi(z)$$

is called the *kernel function* associated with the feature map Φ. Such kernel functions have the property that for any $x_1, \ldots, x_n$, the matrix K with entries

$$K_{ij} = k(x_i, x_j)$$

is positive semidefinite. This turns out to be the key property to define kernel functions. We say a symmetric function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel function if it has this positive semidefiniteness property.

It is clear that positive combinations of kernel functions are kernel functions, since this is also true for positive semidefinite matrices. It is also true that if k_1 and k_2 are kernel functions, then so is $k_1 k_2$. This follows because the elementwise product of two positive semidefinite matrices is positive semidefinite.

Using these rules, we see that the function

$$k(x, z) = (a + b x^T z)^p$$

where $a, b \geq 0$, p a positive integer is a kernel function. Such kernels are called a polynomial kernels. For every polynomial kernel, there exists an associated lifting function Φ with each coordinate of Φ being a monomial such that

$$k(x, z) = \Phi(x)^T \Phi(z) \,.$$

As a simple example, consider the 1-dimensional case of a kernel

$$k(u, v) = (1 + uv)^2 \,.$$

Then $k(u, v) = \Phi(u)^T \Phi(v)$ for

$$\Phi(u) = \begin{bmatrix} 1 \\ \sqrt{2}u \\ u^2 \end{bmatrix} \,.$$

We can generalize polynomial kernels to *Taylor Series* kernels. Suppose that the one dimensional function h has a convergent Taylor series for all $t \in [-R, R]$:

$$h(t) = \sum_{j=1}^{\infty} a_j t^j$$

where $a_j \geq 0$ for all j. Then the function

$$k(x, z) = h(\langle x, z \rangle)$$

is a positive definite kernel. This follows because each term $\langle x, z \rangle^j$ is a kernel, and we are taking a nonnegative combination of these polynomial kernels. The feature space of this kernel is the span of the monomials of degrees where the a_j are nonzero.

Two example kernels of this form are the *exponential kernel*

$$k(x, z) = \exp(\gamma \langle x, z \rangle)$$

and the *arcsine kernel*

$$k(x, z) = \sin^{-1}(\langle x, z \rangle),$$

which is a kernel for x, z on the unit sphere.

Another important kernel is the Gaussian kernel:

$$k(x, z) = \exp\left(-\tfrac{\gamma}{2}\|x - z\|^2\right).$$

The Gaussian kernel can be thought of as first lifting data using the exponential kernel then projecting onto the unit sphere in the lifted space.

We note that there are many kernels with the same feature space. Any Taylor Series kernel with positive coefficients will have the same set of features. The feature space associated Gaussian kernel is equivalent to the span of radial basis functions. What distinguishes the kernels beyond the features they represent? The key is to look at the norm. Suppose we want to find a fit of the form

$$f(x_j) = w^T \Phi(x_j) \qquad \text{for } j = 1, 2, \ldots, n.$$

In the case when Φ maps into a space with more dimensions than the number of data points we have acquired, there will be an infinite number of w vectors that perfectly interpolate the data. As we saw in our introduction to supervised learning, a convenient means to pick an interpolator is to choose the one with smallest norm. Let's see how the norm interacts with the form of the kernel. Suppose our kernel is a Taylor series kernel

$$h(t) = \sum_{j=1}^{\infty} a_j \langle x, z \rangle^j.$$

Then the smaller a_j, the larger the corresponding w_j should have to be. Thus, the a_j in the kernel expansion govern how readily we allow each feature in a least-norm fit. If we consider the exponential kernel with parameter γ, then $a_j = \frac{1}{j!}\gamma^j$. Hence, for large values of γ, only low degree terms will be selected. As we decrease γ, we allow for higher degree terms to enter the approximation. Higher degree terms tend to be more sensitive to perturbations in the data than lower degree ones, so γ should be set as large as possible while still providing desirable predictive performance.

The main appeal of working with kernel representations is they translate into simple algorithms with bounded complexity. Since we restrict our attention to functions in the span of the data, our functions take the form

$$f(x) = \left(\sum_i \alpha_i \Phi(x_i)^T\right) \Phi(x) = \sum_i \alpha_i k(x_i, x)\,.$$

We can thus pose all of our optimization problems in terms of the coefficients α_i. This means that any particular problem will have at most n parameters to search for. Even the norm of f can be computed without ever explicitly computing the feature embedding. Recall that when $f(x) = w^T\Phi(x)$ with $w = \sum_i \alpha_i \Phi(x_i)$, we have

$$\|w\|^2 = \left\|\sum_i \alpha_i \Phi(x_i)\right\|^2 = \alpha^T K \alpha\,,$$

where K is the matrix with ijth entry $k(x_i, x_j)$. As we will see in the optimization chapter, such representations turn out to be optimal in most machine learning problems. Functions learned by ERM methods on kernel spaces are weighted sums of the similarity (dot product) between the training data and the new data. When k is a Gaussian kernel, this relationship is even more evident: The optimal solution is simply a radial basis function whose anchor points are given by the training data points.

Neural networks

Though the name originates from the study of neuroscience, modern neural nets arguably have little to do with the brain. Neural nets are mathematically a composition of differentiable functions, typically alternating between componentwise nonlinearities and linear maps. The simplest example of a neural network would be

$$f(x) = w^T\sigma(Ax + b)$$

Where w and b are vectors, A is a matrix, and σ is a componentwise nonlinearity, applying the same nonlinear function to each component of its input.

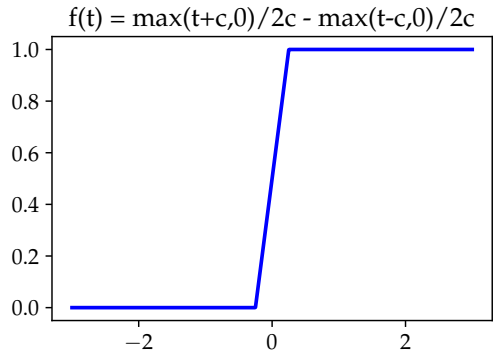

Figure 3: Creating a step function from ReLUs. Here, c=1/4.

The typically used nonlinearities are not Gaussian bumps. Indeed, until recently most neural nets used *sigmoid nonlinearities* where $\sigma(t)$ is some function that is 0 at negative infinity, 1 at positive infinity, and strictly increasing. Popular choices of such functions include $\sigma(t) = \tanh(t)$ or $\sigma(t) = \frac{1}{\exp(-t)+1}$. More recently, another nonlinearity became overwhelmingly popular, the *rectified linear unit* or ReLU:

$$\sigma(t) = \max\{t, 0\}$$

This simple nonlinearity is easy to implement and differentiate in hardware.

Though these nonlinearities are all different, they all generate similar function spaces that can approximate each other. In fact, just as was the case with kernel feature maps, neural networks are powerful enough to approximate any continuous function if enough bases are used. A simple argument by Cybenko clarifies why only a bit of nonlinearity is needed for universal approximation.[7]

Suppose that we can approximate the unit step function $u(t) = \mathbb{1}\{t > 0\}$ as a linear combination of shifts of $\sigma(t)$. A sigmoidal function like tanh is already such an approximation and $\tanh(\alpha t)$ converges to the unit step as α approaches ∞. For ReLU functions we have for any $c > 0$ that

$$\frac{1}{2c}\max\{t+c, 0\} - \frac{1}{2c}\max\{t-c, 0\} = \begin{cases} 0 & t < -c \\ 1 & t > c \\ \frac{t+c}{2c} & \text{otherwise} \end{cases}.$$

It turns out that approximation of such step functions is all that is needed for universal approximation.

To see why, suppose we have a nonzero function g that is not well-

approximated by a sum of the form

$$\sum_{i=1}^{N} w_i \sigma(a_i^T x + b_i)\,.$$

This means that we must have a nonzero function f that lies outside the span of sigmoids. We can take f to be the projection of g onto the orthogonal complement of the span of the sigmoidal functions. This function f will satisfy

$$\int \sigma(a^T x + b) f(x) \mathrm{d}x = 0$$

for all vectors a and scalars b. In turn, since our nonlinearity can approximate step functions, this implies that for any a and any t_0 and t_1,

$$\int \mathbb{1}\{t_0 \leq a^T x \leq t_1\} f(x) \mathrm{d}x = 0\,.$$

We can approximate any continuous function as a sum of the indicator function of intervals, which means

$$\int h(a^T x) f(x) \mathrm{d}x = 0$$

for any continuous function h and any vector a. Using $h(t) = \exp(it)$ proves that the Fourier transform of f is equal to zero, and hence that f itself equals zero. This is a contradiction.

The core of Cybenko's argument is a reduction to approximation in one dimension. From this perspective, it suffices to find a nonlinearity that can well-approximate "bump functions" which are nearly equal to zero outside a specified interval and are equal to 1 at the center of the interval. This opens up a variety of potential nonlinearities for use as universal approximators.

While elegant and simple, Cybenko's argument does not tell us *how many* terms we need in our approximation. More refined work on this topic was pursued in the 1990s. Barron[8] used a similar argument about step functions combined with a powerful randomized analysis due to Maurey.[9] Similar results were derived for sinusoids[10] by Jones and ReLU networks[11] by Breiman. All of these results showed that two-layer networks sufficed for universal approximation, and quantified how the number of basis functions required scaled with respect to the complexity of the approximated function.

Random features

Though the idea seems a bit absurd, a powerful means of choosing basis function is by random selection. Suppose we have a parametric family

of basis functions $b(x;\vartheta)$. A random feature map chooses $\vartheta_1, \ldots, \vartheta_D$ from some distribution on ϑ, and use the feature map

$$\Phi_{\text{rf}}(x) = \begin{bmatrix} b(x;\vartheta_1) \\ b(x;\vartheta_2) \\ \vdots \\ b(x;\vartheta_D) \end{bmatrix}.$$

The corresponding prediction functions are of the form

$$f(x) = \sum_{k=1}^{D} w_k b(x;\vartheta_k)$$

which looks very much like a neural network. The main difference is a matter of emphasis: here we are stipulating that the parameters ϑ_k are random variables, whereas in neural networks, ϑ_k would be considered parameters to be determined by the particular function we aim to approximate.

Why might such a random set of functions work well to approximate complex functional behavior? First, from the perspective of optimization, it might not be too surprising that a random set of nonlinear basis functions will be linearly independent. Hence, if we choose enough of them, we should be able to fit any set of desired labels.

Second, random feature maps are closely connected with kernel spaces. The connections were initially drawn out in work by Rahimi and Recht.[12,13] Any random feature map generates an empirical kernel, $\Phi_{\text{rf}}(x)^T\Phi_{\text{rf}}(z)$. The expected value of this kernel can be associated with some Reproducing Kernel Hilbert Space.

$$\begin{aligned}\mathbb{E}[\tfrac{1}{D}\Phi_{\text{rf}}(x)^T\Phi_{\text{rf}}(z)] &= \mathbb{E}\left[\frac{1}{D}\sum_{k=1}^{D} b(x;\vartheta_k)b(z;\vartheta_k)\right] \\ &= \mathbb{E}[b(x;\vartheta_1)b(z;\vartheta_1)] = \int p(\vartheta)b(x;\vartheta)b(z;\vartheta)\mathrm{d}\vartheta\end{aligned}$$

In expectation, the random feature map yields a kernel given by the final integral expressions. There are many interesting kernels that can be written as such an integral. In particular, the Gaussian kernel would arise if

$$\begin{aligned}p(\vartheta) &= \mathcal{N}(0,\gamma I) \\ b(x;\vartheta) &= [\cos(\vartheta^T x),\ \sin(\vartheta^T x)]\end{aligned}$$

To see this, recall that the Fourier transform of a Gaussian is a Gaussian,

and write:

$$
\begin{aligned}
& k(x,z) \\
& = \exp(-\tfrac{\gamma}{2}\|x-z\|^2) \\
& = \frac{1}{(2\pi\gamma)^{d/2}} \int e^{-\frac{\|v\|^2}{2\gamma}} \exp(iv^T(x-z))\mathrm{d}v \\
& = \frac{1}{(2\pi\gamma)^{d/2}} \int e^{-\frac{\|v\|^2}{2\gamma}} \left\{\cos(v^Tx)\cos(v^Tz) + \sin(v^Tx)\sin(v^Tz)\right\} \mathrm{d}v\,.
\end{aligned}
$$

This calculation gives new insights into the feature space associated with a Gaussian kernel. It shows that the Gaussian kernel is a continuous mixture of inner products of sines and cosines. The sinusoids are weighted by a Gaussian function on their frequency: high frequency sinusoids have vanishing weight in this expansion. The parameter γ controls how quickly the higher frequencies are damped. Hence, the feature space here can be thought of as low frequency sinusoids. If we sample a frequency from a Gaussian distribution, it will be low frequency (i.e., have small norm) with high probability. Hence, a random collection of low frequency sinusoids approximately spans the same space as that spanned by a Gaussian kernel.

If instead of using sinusoids, we chose our random features to be $\mathrm{ReLU}(v^Tx)$, our kernel would become

$$
\begin{aligned}
k(x,z) &= \frac{1}{(2\pi)^{d/2}} \int \exp\left(-\frac{\|v\|^2}{2}\right) \mathrm{ReLU}(v^Tx)\,\mathrm{ReLU}(v^Tz)\mathrm{d}v \\
&= \|x\|\|z\| \left\{\sin(\vartheta) + (\pi-\vartheta)\cos(\vartheta)\right\}\,,
\end{aligned}
$$

where

$$
\vartheta = \cos^{-1}\left(\frac{\langle x,z\rangle}{\|x\|\|z\|}\right)\,.
$$

This computation was first made by Cho and Saul.[14] Both the Gaussian kernel and this "ReLU kernel" are universal Taylor kernels, and, when plotted, we see even are comparable on unit norm data.

Prediction functions built with random features are just randomly wired neural networks. This connection is useful for multiple reasons. First, as we will see in the next chapter, optimization of the weights w_k is far easier than joint optimization of the weights and the parameters ϑ. Second, the connection to kernel methods makes the generalization properties of random features straightforward to analyze. Third, much of the recent theory of neural nets is based on connections between random feature maps and the randomization at initialization of neural networks. The main drawback of random features is that, in practice, they often require large dimensions before they provide predictions on par with neural networks.

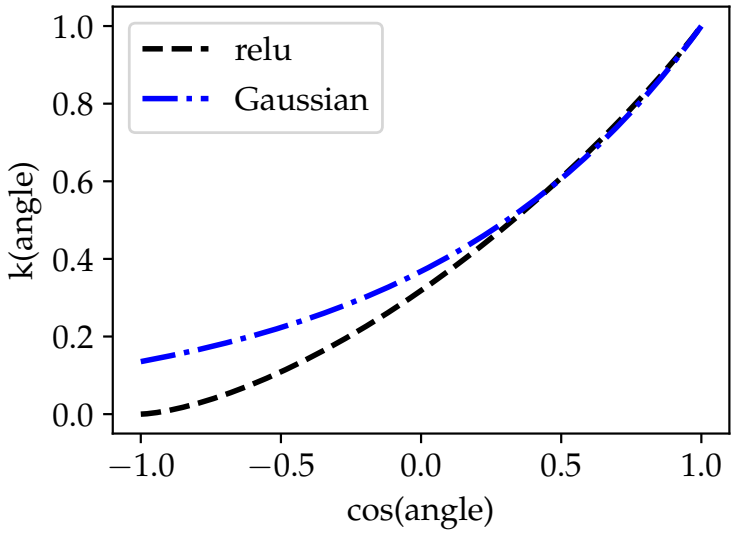

Figure 4: Comparison of the Gaussian and Arccosine Kernels. Plotting the kernel value as a function of the angle between two unit norm vectors.

These tradeoffs are worth considering when designing and implementing nonlinear prediction functions.

Returning to the radial basis expansion

$$f(x) = \sum_{j=1}^{k} w_j e^{-\gamma \|x - z_j\|^2},$$

we see that this expression could be a neural network, a kernel machine, or a random feature network. The main distinction between these three methods is how the z_j are selected. Neural nets search require some sort of optimization procedure to find the z_j. Kernel machines place the z_j on the training data. Random features select z_j at random. The best choice for any particular prediction problem will always be dictated by the constraints of practice.

Chapter notes

To our knowledge, there is no full and holistic account of measurement and quantization, especially when quantization is motivated by data science applications. From the statistical signal processing viewpoint, we have a full and complete theory of quantization in terms of understanding what signals can be reconstructed from digital measurements. The Nyquist-Shannon

theory allows us to understand what parts of signals may be lost and what artifacts are introduced. Such theory is now canon in undergraduate courses on signal processing. See, e.g., Oppenheim and Willsky.[15] For task-driven sampling, the field remains quite open. The theory of compressive sensing led to many recent and exciting developments in this space, showing that task-driven sampling where we combined domain knowledge, computation, and device design could reduce the data acquisition burdens of many pattern recognition tasks.[16] The theory of experiment design and survey design has taken some cues from task-driven sampling.

Reproducing kernels have been used in pattern recognition and machine learning for nearly as long as neural networks. Kernels and Hilbert spaces were first used in time series prediction in the late 1940s, with fundamental work by Karhunen–Loève showing that the covariance function of a time series was a Mercer Kernel.[17,18] Shortly thereafter, Reproducing Kernel Hilbert Spaces were formalized by Aronszajn in 1950.[19] Parzen was likely the first to show that time series prediction problem could be reduced to solving a least-squares problem in an RKHS and hence could be computed by solving a linear system.[20] Wahba's survey of RKHS techniques in statistics covers many other developments post-Parzen.[21] For further reading on the theory and application of kernels in machine learning consult the texts by Schölkopf and Smola[22] and Shawe-Taylor and Cristianini.[23]

Also since its inception, researchers have been fascinated by the approximation power of neural networks. Rosenblatt discussed properties of universal approximation in his monograph on neurodynamics.[24] It was in the 80s when it became clear that though neural networks were able to approximate any continuous function, they needed to be made more complex and intricate in order to achieve high quality approximation. Cybenko provided a simple proof that neural nets were dense in the space of continuous functions, though did not estimate how large such networks might need to be.[7] An elegant, randomized argument by Maurey[9] led to a variety of approximation results which quantified how many basis terms were needed. Notably, Jones showed that a simple greedy method could approximate any continuous function with a sum of sinusoids.[10] Barron shows that similar greedy methods could be used[8] to build neural nets that approximated any function. Breiman analyzed ReLU networks using the same framework.[11] The general theory of approximation by bases is rich, and a Pinkus' book details some of the necessary and sufficient conditions to achieve high quality approximations with as few bases as possible.[25]

That randomly wired neural networks could solve pattern recognition problems also has a long history. Minsky's first electronic neural network, SNARC, was randomly wired. The story of SNARC (Stochastic Neural Analog Reinforcement Calculator) is rather apocryphal. There are no known

photographs of the assembled device, although a 2019 article by Akst has a picture of one of the "neurons".[26] The commonly referenced publication, a Harvard technical report, appears to not be published. However, the "randomly wired" story lives on, and it is one that Minsky told countless times through his life. Many years later, Rahimi and Recht built upon the approximation theory of Maurey, Barron, and Jones to show that random combinations of basis functions could approximate continuous functions well, and that such random combinations could be thought of as approximately solving prediction problems in a RKHS.[13,12,27] This work was later used as a means to understand neural networks, and, in particular, the influence of their random initializations. Daniely et al. computed the kernel spaces associated with that randomly initialized neural networks,[28] and Jacot et al. pioneered a line of work using kernels to understand the dynamics of neural net training.[29]

There has been noted cultural tension between the neural-net and kernel "camps." For instance, the tone of the introduction of work by Decoste and Schölkopf telegraphs a disdain by neural net proponents of the Support Vector Machine.[30]

> Initially, SVMs had been considered a theoretically elegant spin-off of the general but, allegedly, largely useless VC-theory of statistical learning. In 1996, using the first methods for incorporating prior knowledge, SVMs became competitive with the state of the art in the handwritten digit classification benchmarks that were popularized in the machine learning community by AT&T and Bell Labs. At that point, practitioners who are not interested in theory, but in results, could no longer ignore SVMs.

With the rise of deep learning, however, there are a variety of machine learning benchmarks where SVMs or other kernel methods fail to match the performance of neural networks. Many have dismissed kernel methods as a framework whose time has past. However, kernels play an evermore active role in helping to better understand neural networks and insights from deep learning have helped to improve the power of kernel methods on pattern recognition tasks.[31] Neural nets and kernels are complementary, and active research in machine learning aims to bring these two views more closely together.

Bibliography

[1] David J. Hand. *Measurement Theory and Practice: The World Through Quantification*. Wiley, 2010.

[2] David J. Hand. *Measurement: A Very Short Introduction*. Oxford University Press, 2016.

[3] Deborah L. Bandalos. *Measurement Theory and Applications for the Social Sciences*. Guilford Publications, 2018.

[4] Lisa Gitelman. *Raw Data Is an Oxymoron*. MIT Press, 2013.

[5] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica*, May 2016.

[6] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78, 2018.

[7] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[8] Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *Transactions on Information Theory*, 39(3):930–945, 1993.

[9] Gilles Pisier. Remarques sur un résultat non publié de B. Maurey. In *Séminaire d'analyse fonctionnelle*. Ecole Polytechnique Centre de Mathematiques, 1980-1981.

[10] Lee K. Jones. A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. *Annals of Statistics*, 20(1):608–613, 1992.

[11] Leo Breiman. Hinging hyperplanes for regression, classification, and function approximation. *Transactions on Information Theory*, 39(3):999–1013, 1993.

[12] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

[13] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*, 2008.

[14] Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, 2009.

[15] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals and Systems*. Prentice-Hall International, 1997.

[16] Emmanuel J. Candès and Michael B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[17] Kari Karhunen. Über lineare Methoden in der Wahrscheinlichkeitsrechnung. *Annales Academia Scientiarum Fennica Mathematica, Series A*, (37):1–47, 1947.

[18] Michel Loève. Functions aleatoire de second ordre. *Revue Science*, 84:195–206, 1946.

[19] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[20] Emmanuel Parzen. An approach to time series analysis. *The Annals of Mathematical Statistics*, 32(4):951–989, 1961.

[21] Grace Wahba. *Spline Models for Observational Data*. SIAM, 1990.

[22] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[23] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[24] Frank Rosenblatt. *Principles of Neurodynamics: Perceptions and the Theory of Brain Mechanisms*. Spartan, 1962.

[25] Allan Pinkus. *N-Widths in Approximation Theory*. Springer, 1985.

[26] Jef Akst. Machine, learning, 1951. *The Scientist*, May 2019.

[27] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *Allerton Conference on Communication, Control, and Computing*, 2008.

[28] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances in Neural Information Processing Systems*, 2016.

[29] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, pages 8580–8589, 2018.

[30] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[31] Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Jonathan Ragan-Kelley, Ludwig Schmidt, and Benjamin Recht. Neural kernels without tangents. In *International Conference on Machine Learning*, 2020.