Features and Basis Functions

Ryan P. Adams
COS 324 – Elements of Machine Learning
Princeton University

At this point, you might be reasonably wondering how far we can really get with linear regression. The world is a complicated place, and we can't expect linear models to capture the wide variety of functions that we might need to approximate in supervised learning. Moreover, linear regression models seem to depend on the inputs being vectors in $\mathbb{R}^D$ and many data don't really map easily to real-valued vectors. For example, how could we predict the properties of a molecule with linear regression? A molecule is some complicated graph-like object with atoms and bonds – definitely not something living in $\mathbb{R}^D$. The answer is to introduce *features* or *basis functions* that make turn various kinds of inputs into numerical vectors.

Making Linear Regression Nonlinear

Let's start off by just thinking about how to make regression nonlinear in the least squares case. As before, let's start off with the simple univariate case where we have some data $\{x_n, y_n\}_{n=1}^N$ with $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$. We modeled these data with a line in slope-intercept form: $y = mx + b$ with unknown m and b. Then later we observed that we could add an extra constant-valued feature to our data and now not have to treat b as a separate kind of parameter; we put both m and b into a single vector parameter $\boldsymbol{w}$. Let's slightly change the way that we think about appending a vector. Rather than it being a trick that we apply, let's formalize it as a vector valued function $\Phi_{\mathsf{lin}} : \mathbb{R} \to \mathbb{R}^2$ where

$$\phi_0(x) = 1 \qquad\qquad \phi_1(x) = x \,. \tag{1}$$

When we write out the parameterized straight lines we get

$$y = \boldsymbol{w}^\mathsf{T} \Phi_{\mathsf{lin}}(x) = \phi_0(x) w_0 + \phi_1(x) w_1 = w_0 + w_1 x \,. \tag{2}$$

There's nothing new here except we've replaced the idea of "append a 1 to the features" with "transform the data using a function Φ". Note that what this let us do is write y as a degree-1 polynomial. What if we made it a degree-2 polynomial instead? The transformation would now have 3 terms:

$$\phi_0(x) = 1 \qquad\qquad \phi_1(x) = x \qquad\qquad \phi_2(x) = x^2 \tag{3}$$

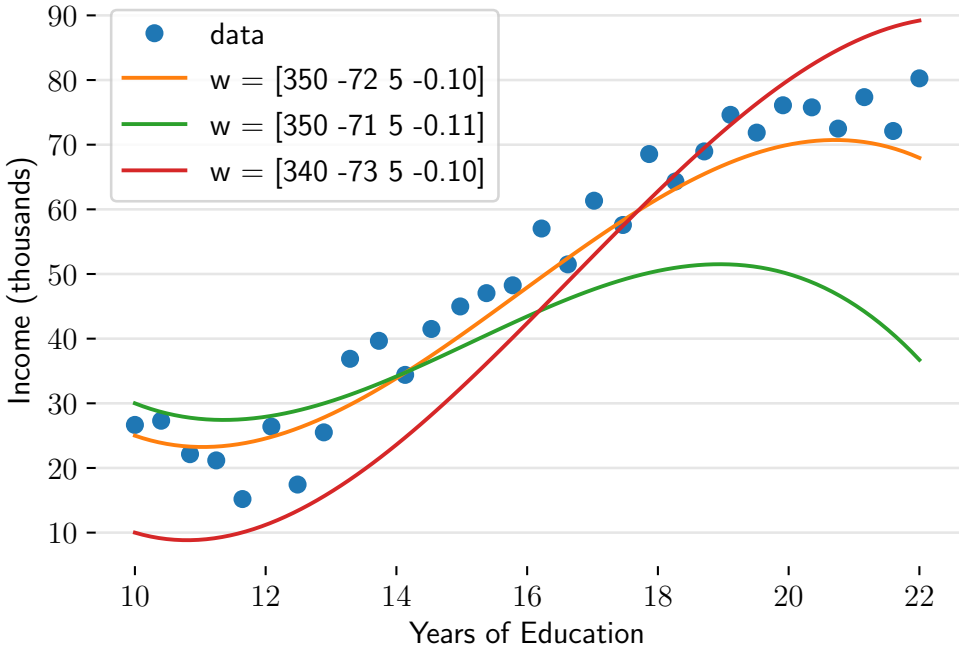


Figure 1: Three cubic polynomials roughly fitting the synthetic `Income` data.

and we'd have

$$y = \boldsymbol{w}^\mathsf{T}\Phi_{\text{quad}}(x) = \phi_0(x)w_0 + \phi_1(x)w_1 + \phi_2(x)w_2 = w_0 + w_1x + w_2x^2\,. \tag{4}$$

We could of course go farther and make it a cubic polynomial with $\Phi_{\text{cub}}(x) = [1, x, x^2, x^3]^\mathsf{T}$:

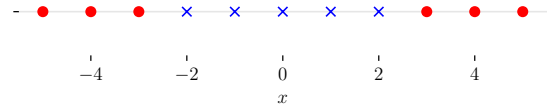
$$y = \boldsymbol{w}^\mathsf{T}\Phi_{\text{cub}}(x) = \phi_0(x)w_0 + \phi_1(x)w_1 + \phi_2(x)w_2 + \phi_3(x)w_3 = w_0 + w_1x + w_2x^2 + w_3x^3\,. \tag{5}$$

Some cubic polynomials are shown with the `Income` data in Figure 1.

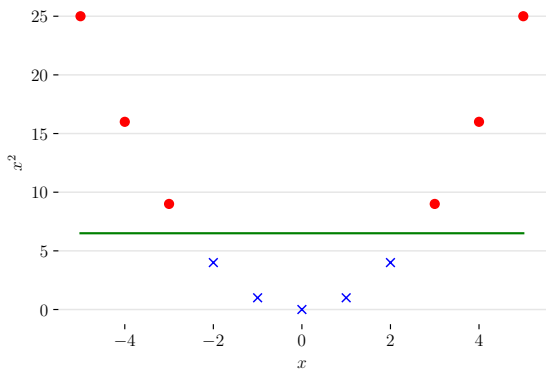
Recall that when we added the constant term to our features, nothing really changed about how we did the math. Interestingly, nothing changes here, either, due to adding a quadratic term. We are just now estimating a larger vector $\boldsymbol{w}$. Approaching this more generally, if we have some input space $\mathcal{X}$ then our transformation is a function $\Phi : \mathcal{X} \to \mathbb{R}^J$ that takes those inputs and turns them into vectors in $\mathbb{R}^J$. When we do regression, rather than writing $\boldsymbol{X}$ for our design matrix, we write $\boldsymbol{\Phi}$ for the design matrix after transforming the data, i.e.,

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \cdots & \phi_{J-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \cdots & \phi_{J-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \cdots & \phi_{J-1}(\boldsymbol{x}_N) \end{bmatrix}. \tag{6}$$

(I'm using zero indexing here to stay consistent with the polynomial setup from before where $\phi_0(\boldsymbol{x}) = 1$ but that is not a crucial property of this setup.) Everything else is the same as in the simple linear



(a) A simple data set in one dimension with two classes. These data are not separable with a single threshold.



(b) Adding a transformation that includes an x^2 term makes the data linearly separable (green line).

Figure 2: Example of using basis functions in classification

regression case, just with $\mathbf{\Phi}$ instead of $\boldsymbol{X}$. The objective function for least squares regression is now:

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} \left\{ \frac{1}{N} (\mathbf{\Phi}\boldsymbol{w} - \boldsymbol{y})^\mathsf{T} (\mathbf{\Phi}\boldsymbol{w} - \boldsymbol{y}) \right\} . \tag{7}$$

Assuming $N \geq J$, the solution is going to be

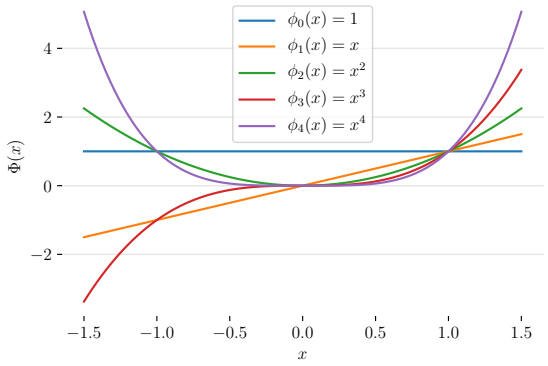
$$\boldsymbol{w}^\star = (\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathsf{T}\boldsymbol{y} . \tag{8}$$

Since we are fixing Φ, we could treat this as a linear problem. But now the solution *as a function of* $\boldsymbol{x}$ is nonlinear. We refer to these ϕ_j as *basis function* or *derived features*.

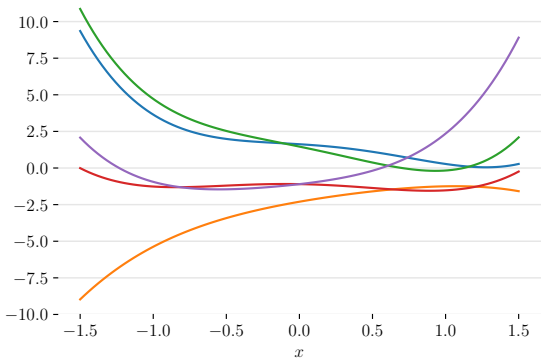
We can gain some further intuition about this by thinking about classification. We haven't talked much about it yet, but a key concept in classification is the idea of the decision boundary, where we transition from making predictions of one category (e.g., *dogs*) into a different category (e.g., *cats*). We want our decision boundaries to be simple – ideally linear, but of course not all data can be categorized with simple decision boundaries. Figure 2a shows a very simple data set with two classes. In this figure there is no way to put down a single threshold that perfectly separates the two classes. However, if we augment the features with x^2 suddenly a simple linear threshold can classify the data, as shown by the green line in Figure 2b.

Examples of Basis Functions

Polynomials In the previous section, we introduced quadratic and cubic polynomials in one dimension as an example of basis functions in linear regression. Figure 3a shows this basis out to x^4 and Figure 3b is five random weightings of these basis functions to give an idea of what kinds of functions it can produce. We can generalize this to multidimensional $\mathcal{X}$ in different ways. One



(a) Polynomial basis out to degree 4.



(b) Five random weightings of the basis functions.

Figure 3: Basic polynomial basis with random functions to show typical examples

option is to construct a polynomial with all possible products of the dimensions out to some degree, although this will tend to get large very fast. Another option is to just expand each dimension out in single-variable polynomials and combine them without interaction terms (or with low degree interactions).

Fourier Series Another reasonable option, particularly for periodic data or data with known boundaries, is the Fourier basis. One straightforward approach is to choose a set of frequencies and phases and construct the basis as in:

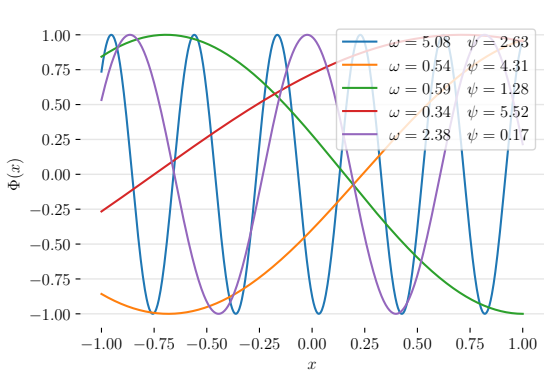
$$\phi_0(x) = 1 \qquad \phi_j(x) = \cos(\omega_j x + \psi_j) \quad \text{for } j > 0\,. \tag{9}$$

Figure 4a shows five functions with arbitrarily chosen ω_j and ψ_j. Figure 4b shows five random weightings of these functions to show typical regressions. A general approach for regression on the interval $[-1, 1]$ is to use a truncated Fourier series:

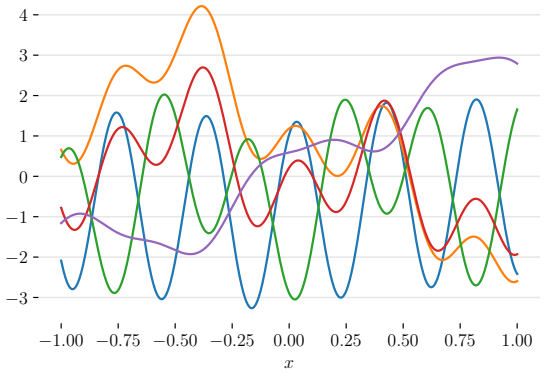
$$\phi_j(x) = \begin{cases} 1 & \text{if } j = 0 \\ \cos(\pi x j) & j \text{ odd} \\ \sin(\pi x j) & j \text{ even} \end{cases}. \tag{10}$$

You may recall from calculus that one can approximate any sufficiently well-behaved function on an interval arbitrarily well with a Fourier series. Thus it might be a nice choice of basis for linear regression. Figure 5a shows five elements of the truncated Fourier basis, out to $\omega = 2$. Figure 5b shows five random weightings of this basis. Although out of scope for this course, this basis can be made multidimensional by treating each dimension with its own series, by taking products between terms, and also by applying a set of linear transformations to the input features before using them as the argument to the cosines and sines. 5This last approach, with a set of random weights and phases turns out to be a very powerful way to do regression and classification and is sometimes called a *randomized Fourier basis*[1].

[1]See *Random Features for Large Scale Kernel Machines* by Ali Rahimi and Ben Recht in NIPS 2007.



(a) Cosine basis example with several arbitrary frequencies and phases.



(b) Five random weightings of the basis functions shown on the left.

Figure 4: Example of cosine basis with random functions to show typical functions

Piecewise Linear One flexible way to construct a regression basis is to divide $\mathcal{X}$ into disjoint regions and then allow each of these to have its own (local) linear regressor. This can be seen as basis function linear regression. In one dimension, imagine disjoint intervals $\mathcal{X}_1, \ldots, \mathcal{X}_K$, and basis functions

$$\phi_{2k-1}(x) = \mathbf{1}_{\mathcal{X}_k}(x) \qquad \phi_{2k}(x) = x\mathbf{1}_{\mathcal{X}_k}(x)\,. \tag{11}$$

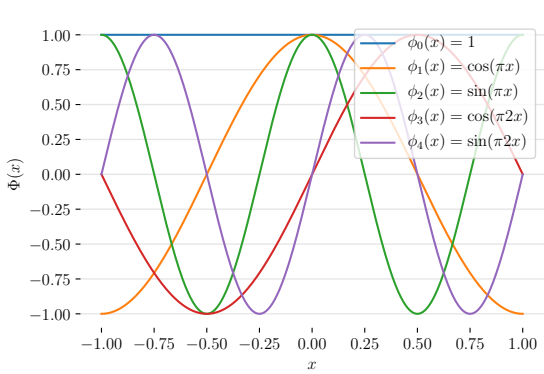
These are zero when x is not in the relevant interval, and linear functions when they are. Figure 6a shows some locally-linear basis functions on length-one intervals, as can be seen by the different colors. Figure 6b shows random weightings of these functions to give an idea of what kind of functions can be modeled. This could also be combined with other basis functions, or additional constraints could be added to ensure continuity. It can also be directly generalized to multiple dimensions by combining multiple regression with indicator functions.

Radial Basis Functions A common kind of basis function in practice is a *radial basis function* (RBF). A radial basis function is a function that is symmetric around a point and then typically decays to zero as you get farther from the center. One of the most common is an exponentiated quadratic (basically a scaled isotropic Gaussian):

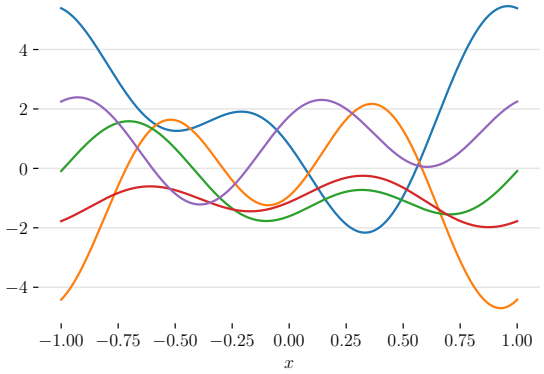
$$\phi_j(\boldsymbol{x}) = \exp\left\{-\frac{1}{\ell}||\boldsymbol{x} - \boldsymbol{\mu}_j||_2^2\right\} = \exp\left\{-\frac{1}{2\ell}\sum_{d=1}^{D}(x_d - \mu_{j,d})^2\right\}\,. \tag{12}$$

This is a bump centered at $\boldsymbol{\mu}_j$ and the parameter $\ell > 0$ determines the (squared) length scale of the decay. Figure 7a shows several radial basis functions and Figure 7b shows five random weightings. There are other types of radial basis functions as well that vary in shape and how fast they fall off. Another example is:

$$\phi_j(\boldsymbol{x}) = \frac{1}{1 + \frac{1}{\ell}||\boldsymbol{x} - \boldsymbol{\mu}_j||_2^2}\,, \tag{13}$$



(a) Fourier basis out to $\omega = 2$.



(b) Five random weightings of the basis functions shown on the left.

Figure 5: Example of truncated Fourier basis and random functions to show typical functions

which is a scaled Cauchy distribution, and

$$\phi_j(\boldsymbol{x}) = \exp\left\{-\frac{1}{\ell}||\boldsymbol{x} - \boldsymbol{\mu}_j||_2\right\}, \tag{14}$$

which is a scaled Laplace distribution (note that the Euclidean distance isn't squared).

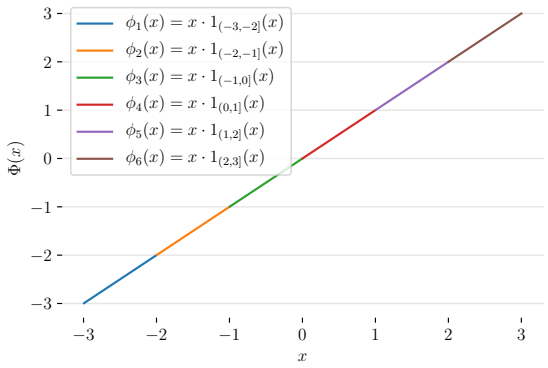
Parameterized Soft Thresholds An idea that will come up again with neural networks is to have the basis functions be soft linear threshold functions:

$$\phi_j(\boldsymbol{x}) = \frac{1}{1 + \exp\left\{-\boldsymbol{x}^\top \boldsymbol{w}_j\right\}}. \tag{15}$$

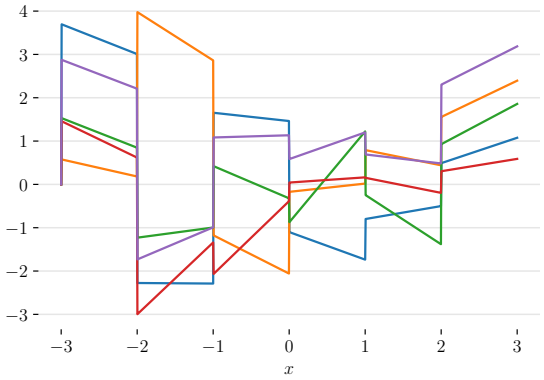
This function is close to one when $\boldsymbol{x}^\top \boldsymbol{w}_j$ is big and positive and becomes close to zero when it is large and negative. A closely related function is the hyperbolic tangent, which goes to -1 instead of zero. Figure 8a shows several tanh functions with different scaling and shifting of the input. Figure 8b shows five random weightings of these basis functions to provide an idea of typical regressors. Neural networks are often composed of basis functions like these where the $\boldsymbol{w}_j$ are learned along with the linear regression weights, using the chain rule.

Engineering Vector Representations

One of the interesting things about the basis function idea is that it gives us flexibility in what kinds of things $\mathcal{X}$ can be. Lots of data that we'd like to use for supervised learning either don't come as vectors, or those vectors aren't very directly useful.



(a) Piecewise linear basis shown as colors in each unit-length segment. Constant terms are not shown.



(b) Five random weightings of the basis functions shown on the left.

Figure 6: Example of a piecewise linear basis and random functions to show typical functions

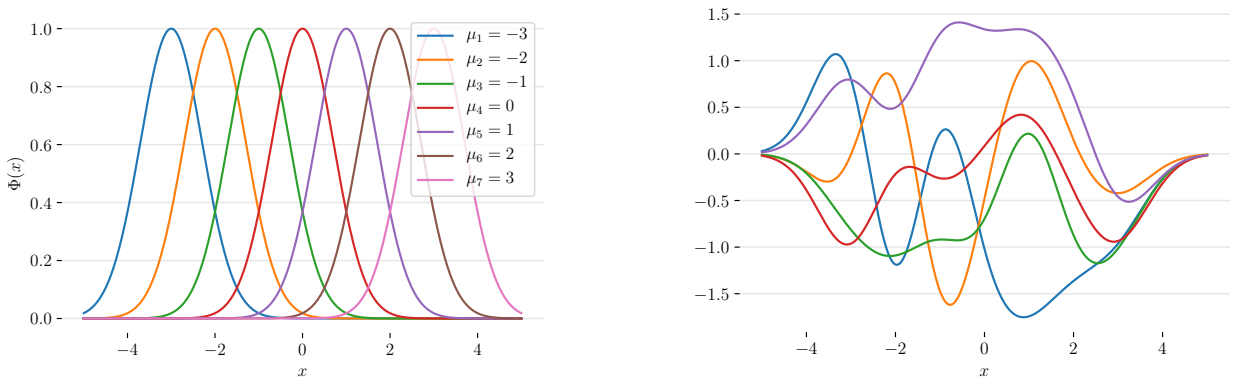
Images Pixels of images are one common example of a situation where the obvious vectorial representation isn't very useful: small translations of an object in a picture don't change the semantic label, but may have a big effect on a "flattened" vector of pixels. As a result, one often uses some kind of basis functions to re-represent images before applying a regressor or classifier. There are many ideas for how to do this. These are outside the scope of this course, but might be worth being aware of in case you have some image data that you'd like to work on.

- *discrete cosine transform (DCT)* – If you think of the single channel of an image as being a two-dimensional function, like a surface, then it's natural to think about representing it via its discrete Fourier transform. The discrete cosine transform essentially does this to re-represent an image, although it throws out the imaginary components. The tremendously successful JPEG format uses the DCT as the basis of its compression.

- *wavelet transform* – The wavelet transform is very similar in concept to the Fourier transform (and DCT), but whereas they are only localized in frequency, wavelets are localized in both space and frequency. So rather than representing an image as its spatial frequencies, wavelets can represent images as spatial frequencies in particular regions. These are appealing both because they work well and because they are interestingly related to perceptual processing that appears to go on in the mammalian visual cortex for things like edge detection. See also *Gabor filters*.

- *histogram of oriented gradients (HOG)* – Strong gradients of intensity in an image generally indicate structure like edges. The HOG descriptor transforms an image into a spatially-localized set of estimates of the presence and direction of these gradients.

- *scale-invariant feature transform (SIFT)* – This is a highly engineered approach to representing images that tries to explicitly deal with invariance to common affine transformations. It



(a) Examples of Gaussian-type radial basis functions.

(b) Five random weightings of the basis functions shown on the left.

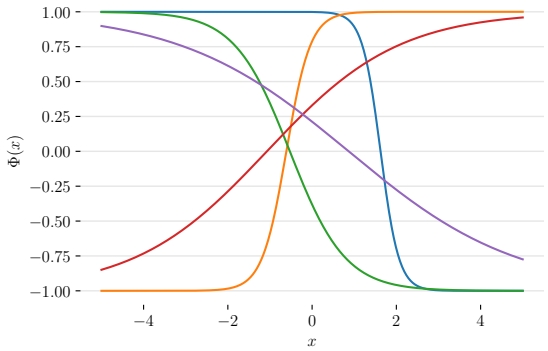
Figure 7: Example of Gaussian-type radial basis functions and a random weighting to show typical functions.

was one of the dominant approaches to computer vision before the resurgence of interest in convolutional neural networks.

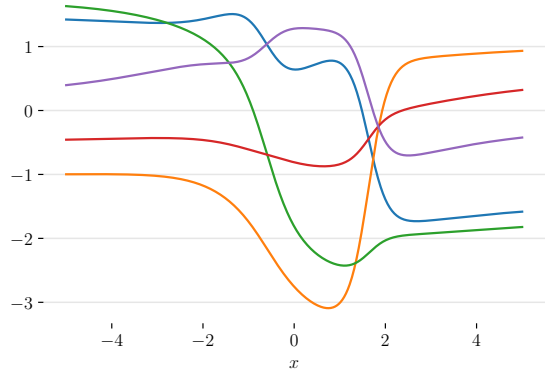
- *GIST descriptors* – Rather than capturing fine-grained local structure like SIFT and HOG, the GIST descriptor tries to capture the global shape present in an image.
- *convolutional neural networks* – These are dominating computer vision right now. These are learned basis functions that are invariant to translations of the image. They often have simple nonlinearities but are stacked to ideally represent image features of increasing complexity.

Speech Raw speech signals are univariate time series representing sound waveforms sampled at some reasonable frequency such as 10kHz. The raw signals are noisy, long and are also undesirably perturbed by temporal translation, so it is difficult to build models around the waveforms directly. Just as in images, it is common to use Fourier-like or wavelet transforms to capture higher-level features than raw waveforms. Other specialized transformations have also emerged over the years, most notably the Mel-frequency cepstrum, which is the discrete cosine transform of a weirdly-scaled local power spectrum. If you look at the details, it is unclear how anyone came up with this representation but it has been successful for decades, only recently being overtaken by convolutional neural network approaches.

Text Natural language text provides a salient example of a kind of object that is difficult to turn into a vector: they are discrete ordered tokens in strings of varying length, with word vocabularies running into the hundreds of thousands or millions – and that's just thinking about one language. For tasks like topic and sentiment analysis, a common thing to do is represent a document by the counts of different words in the vocabulary. If the word "kavanaugh" appears many times in a document, we might think that document is about something different than one where the



(a) Examples of hyperbolic tangent basis functions, with arbitrary scaling and shifting of the input.



(b) Five random weightings of the basis functions shown on the left.

Figure 8: Example of hyperbolic tangent basis functions and a random weighting to show typical functions.

word "microelectronics" appears frequently. For more advanced machine learning tasks, one might extend this to include n-grams, which are sequences of words of length n, e.g., counting bi-grams would track the frequency of phrases like "black sheep" or "tax evasion". This approach becomes challenging when the vocabulary becomes large as the number of n-grams grows exponentially and it's unlikely to have many of them represented in any given document. A more advanced approach to word counts tries to upweight words that are more discriminative and down-weight those that are common. For example, in a sentiment analysis task we might want to pay more attention to "nice", "ugly", and "awful" than to "the", "of", or "today". The approach of TF-IDF, or *term frequencyâĂŞinverse document frequency* is a popular way to make such statistical adjustments when computing a document representation. There are many other ideas to be aware of as well, such as the idea of *word embeddings* in which the vocabulary of a set of documents are mapped to vectorial representations that hopefully capture interesting semantics in its dimensions.

Other Data There are as many different ways to cook up basis functions as there are data. For example, when predicting the properties of molecules, it's common to use so-called *molecular fingerprints* and this is just one idea of out the entire area of cheminformatics. In various kinds of time series, it can be desirable to summarize things into a compact format; in our work on predicting outcomes for ICU patients, we used a switching dynamical system to first segment patient time series into "modes" and then used the durations in those modes as features for a classifier. When representing users in a social network, you might try to base their features on who they connect to, e.g., a user who is followed by MC Hammer might be very different from a user followed by Soledad O'Brien.

Changelog

- 24 September 2018 – Initial version.