Random Fourier Features

Rahimi and Recht's 2007 paper, "Random Features for Large-Scale Kernel Machines", introduces a framework for randomized, low-dimensional approximations of kernel functions. I discuss this paper in detail with a focus on random Fourier features.

PUBLISHED

23 December 2019

Kernel machines

Consider a learning problem with data and targets $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where $\mathbf{x}_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$. Ignoring the bias, a linear model finds a hyperplane $\mathbf{w}$ such that the decision function

$$f^*(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \tag{1}$$

is optimal for some loss function. For example, in logistic regression, we compute the logistic function of $f(\mathbf{x})$, and then threshold the output probability to produce a binary classifier with $\mathcal{Y} = \{0, 1\}$. Obviously, linear models break down when our data are not linearly separable for classification (Figure 1, left) or do not have a linear relationship between the features and targets for regression.

In a *kernel machine* or a *kernel method*, the input domain $\mathcal{X}$ is mapped into another space $\mathcal{V}$ in which the targets may be a linear function of the data. The dimension of $\mathcal{V}$ may be high or even infinite, but kernel methods avoid operating explicitly in this space using the kernel trick: if $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a positive definite kernel, then by Mercer's theorem there exists a *basis function* or *feature map* $\varphi : \mathcal{X} \mapsto \mathcal{V}$ such that

$$k(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle_{\mathcal{V}}. \tag{2}$$

$\langle \cdot, \cdot \rangle_{\mathcal{V}}$ is an inner product in $\mathcal{V}$. If the kernel trick is new to you, please see my previous post. Using the kernel trick and a representer theorem (Kimeldorf & Wahba, 1971), kernel methods construct nonlinear models of $\mathcal{X}$ that are linear in $k(\cdot, \cdot)$,

$$f^*(\mathbf{x}) = \sum_{n=1}^N \alpha_n k(\mathbf{x}, \mathbf{x}_n) = \langle \mathbf{w}, \varphi(\mathbf{x}) \rangle_{\mathcal{V}}. \tag{3}$$

In (3), $f^*(\cdot)$ denotes the optimal $f(\cdot)$. Taken together, (2) and (3) say that provided we have a positive definite kernel function $k(\cdot, \cdot)$, we can avoid operating in the possibly infinite-dimensional space $\mathcal{V}$ and instead only compute over N data points. This works because the optimal decision rule can be expressed as an expansion in terms of the training samples. See (Schölkopf et al., 2001) for a detailed treatment on this topic.

If the representer theorem is new to you, note that you've already seen it in action. For example, the optimal coefficients in linear regression are

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{4}$$

where $\mathbf{X}$ is an $N \times D$ matrix of training data. We make predictions on held out data $\mathbf{X}_*$ as $\mathbf{y}_* = \mathbf{X}_* \hat{\mathbf{w}}$. In other words, the optimal prediction for a linear model can be viewed as an expansion in terms of the training samples. For a second example, recall that the posterior mean of a Gaussian process (GP) regressor is

$$\mathbb{E}[\mathbf{f}_*] = \mathbf{K}_{X_*,X} \overbrace{[\sigma^2 \mathbf{I} + \mathbf{K}_{X,X}]^{-1} \mathbf{y}}^{\beta}, \tag{5}$$

where the notation $\mathbf{K}_{X_*,X}$ denotes the kernel function evaluated at all pairs of samples in $\mathbf{X}_*$ and $\mathbf{X}$. See my previous post on GP regression if needed. For each component in the M-vector $\mathbb{E}[\mathbf{f}_*]$ (each prediction y_m), the GP prediction is a linear-in-$\boldsymbol{\beta}$ model of the kernel evaluated at the test data $\mathbf{x}_m$ against all the training points.

Perhaps the most famous kernel machine is the nonlinear support vector machine (SVM) (Figure 1, right). However, any algorithm that can be represented as a dot product between pairs of samples can be converted into a kernel method using (2). Other methods that can be considered kernel methods are GPs, kernel PCA, and kernel regression.

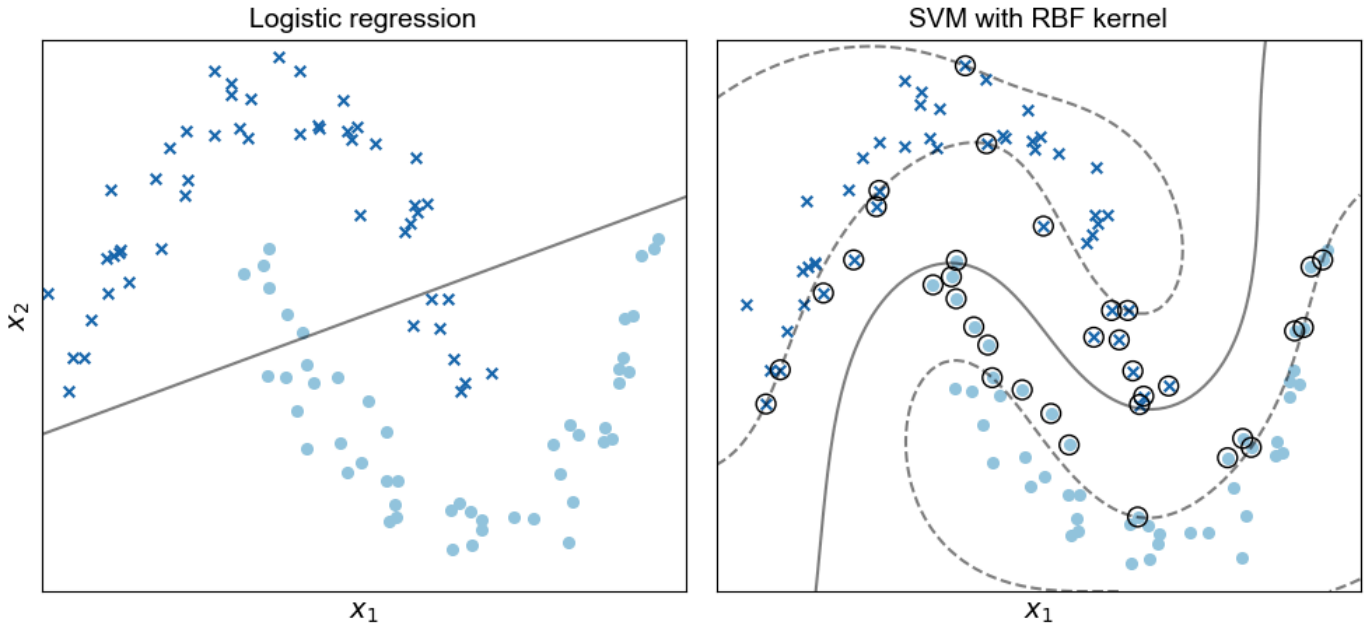

Figure 1. Logistic regression (left) with decision boundary denoted with a solid line and SVM with radial basis function (RBF) kernel (right) on the Scikit-learn half-circles dataset. Support vectors are denoted with circles, and the margins are denoted with dashed lines.

While the kernel trick is a beautiful idea and the conceptual backbone of kernel machines, the problem is that for large datasets (for huge N), the machine must operate on a covariance matrix $\mathbf{K}_X$, induced by a particular kernel function, that is $N \times N$,

$$\mathbf{K}_X = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \tag{6}$$

For example, in the Gaussian process regressor in (5), the right-most covariance kernel $\mathbf{K}_{X,X}$ is an $N \times N$ matrix. To compute $\boldsymbol{\beta}$, we must invert this. Furthermore, to evaluate a test data point, the model must evaluate the sum in (3). While we're avoiding computing in $\mathcal{V}$, we are stuck operating in $\mathbb{R}^N$. In the area of big data, kernel methods do not necessarily scale.

Random features

In their 2007 paper, *Random Features for Large-Scale Kernel Machines* (Rahimi & Recht, 2007), Ali Rahimi and Ben Recht propose a different tack: approximate the above inner product in (2) with a randomized map $\mathbf{z} : \mathbb{R}^D \mapsto \mathbb{R}^R$ where ideally $R \ll N$,

$$k(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle_{\mathcal{V}} \approx \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{y}). \tag{7}$$

See this talk by Rahimi for details on the theoretical guarantees of this approximation. Why does this work and why is it it a good idea? The representer theorem tells us that the optimal solution is a weighted sum of the kernel evaluated at our observations. If we have a good approximation of $\varphi(\cdot)$, then

$$\begin{aligned} f^*(\mathbf{x}) &= \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}) \\ &= \sum_{n=1}^{N} \alpha_n \langle \varphi(\mathbf{x}_n), \varphi(\mathbf{x}) \rangle_{\mathcal{V}} \\ &\approx \sum_{n=1}^{N} \alpha_n \mathbf{z}(\mathbf{x}_n)^\top \mathbf{z}(\mathbf{x}) \\ &= \boldsymbol{\beta}^\top \mathbf{z}(\mathbf{x}). \end{aligned} \tag{8}$$

In other words, provided $\mathbf{z}(\cdot)$ is a good approximation of $\varphi(\cdot)$, then we can simply project our data using $\mathbf{z}(\cdot)$ and then use fast linear models in $\mathbb{R}^R$ rather than $\mathbb{R}^N$ because both $\boldsymbol{\beta}$ and $\mathbf{z}(\cdot)$ are R-vectors. So the task at hand is to find a random projection $\mathbf{z}(\cdot)$ such that it well-approximates the corresponding nonlinear kernel machine.

According to this blog post by Rahimi, this idea was inspired by the following observation. Let $\boldsymbol{\omega}$ be a random D-dimensional vector such that

$$\boldsymbol{\omega} \sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}). \tag{9}$$

Now define h as

$$h : \mathbf{x} \mapsto \exp(i\boldsymbol{\omega}^\top \mathbf{x}). \tag{10}$$

Above, i is the imaginary unit. Let the superscript $*$ denote the complex conjugate. Importantly, recall that the complex conjugate of e^{ix} is e^{-ix}. Then note

$$\begin{aligned} \mathbb{E}_{\boldsymbol{\omega}}[h(\mathbf{x})h(\mathbf{y})^*] &= \mathbb{E}_{\boldsymbol{\omega}}[\exp(i\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))] \\ &= \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) \exp(i\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))\mathrm{d}\boldsymbol{\omega} \\ &= \exp\Big(-\frac{1}{2}(\mathbf{x}-\mathbf{y})^\top(\mathbf{x}-\mathbf{y})\Big). \end{aligned} \tag{11}$$

In other words, the expected value of $h(\mathbf{x})h(\mathbf{y})^*$ is the Gaussian kernel. See A1 for a complete derivation.

This is quite cool, and it is actually a specific instance of a more general result, Bochner's theorem (Rudin, 1962). Quoting Rahimi and Recht's version with small modifications for consistent notation, the theorem is:

> **Bochner's theorem:** A continuous kernel $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$ on $\mathbb{R}^D$ is positive definite if and only if $k(\Delta)$ is the Fourier transform of a non-negative measure.

The Fourier transform of a non-negative measure, call it $p(\omega)$, is

$$k(\boldsymbol{\Delta}) = \int p(\boldsymbol{\omega}) \exp(i\boldsymbol{\omega}\boldsymbol{\Delta})\mathrm{d}\boldsymbol{\omega}. \tag{12}$$

Rahimi and Recht observe that many popular kernels such as the Gaussian (or radial basis function), Laplace, and Cauchy kernels are shift-invariant. The upshot is that the transformation $h(\cdot)$ is an unbiased estimate of $k(\mathbf{x} - \mathbf{y})$. Which kernel is $k(\mathbf{x} - \mathbf{y})$? It depends on the non-negative measure $p(\omega)$.

This gives us a general framework to approximate any shift-invariant kernel by re-defining $h(\cdot)$ in (10) to depend on $\boldsymbol{\omega}$ from any non-negative measure $p(\boldsymbol{\omega})$, not just the spherical Gaussian in (9). Furthermore, if we sample R i.i.d. realizations $\{\boldsymbol{\omega}_r\}_{r=1}^R$, we can lower the variance of this approximation:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{y}) &= k(\mathbf{x} - \mathbf{y}) \\
&= \int p(\boldsymbol{\omega}) \exp(i\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y}))\mathrm{d}\boldsymbol{\omega} \\
&= \mathbb{E}_{\boldsymbol{\omega}}\left[\exp(i\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y}))\right] \\
&\overset{1}{\approx} \frac{1}{R}\sum_{r=1}^{R} \exp(i\boldsymbol{\omega}_r^\top(\mathbf{x} - \mathbf{y})) \\
&= \begin{bmatrix} \frac{1}{\sqrt{R}}\exp(i\boldsymbol{\omega}_1^\top\mathbf{x}) \\ \frac{1}{\sqrt{R}}\exp(i\boldsymbol{\omega}_2^\top\mathbf{x}) \\ \vdots \\ \frac{1}{\sqrt{R}}\exp(i\boldsymbol{\omega}_R^\top\mathbf{x}) \end{bmatrix}^\top \begin{bmatrix} \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{\omega}_1^\top\mathbf{y}) \\ \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{\omega}_2^\top\mathbf{y}) \\ \vdots \\ \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{\omega}_R^\top\mathbf{y}) \end{bmatrix} \\
&\overset{2}{\equiv} \mathbf{h}(\mathbf{x})\mathbf{h}(\mathbf{y})^*.
\end{aligned}
\tag{13}
$$

Step 1 is a Monte Carlo approximation of the expectation. Step 2 is the definition of a random map $\mathbf{h}: \mathbb{R}^D \mapsto \mathbb{R}^R$, so an R-vector of normalized $h(\cdot)$ transformations.

Note that we've talked about the *dot product* $\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{y})$, but above we have $\mathbf{h}(\mathbf{x})\mathbf{h}(\mathbf{y})^*$. As we will see in the next section, the imaginary part of our random map will disappear, and the new transform is what Rahimi and Recht call $\mathbf{z}$.

Fine tuning

Now that we understand the big idea of a low-dimensional, randomized map and why it might work, let's get into the weeds. First, note that since both our distribution $\mathcal{N}_D(\mathbf{0}, \mathbf{I})$ and the kernel $k(\Delta)$ are real-valued, we can write

$$
\begin{aligned}
\exp(i\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y})) &\overset{\dagger}{=} \cos(\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y})) - \cancel{i\sin(\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y}))} \\
&= \cos(\boldsymbol{\omega}^\top(\mathbf{x} - \mathbf{y})).
\end{aligned}
\tag{14}
$$

Step † is Euler's formula. We can then define $z_{\boldsymbol{\omega}}(\mathbf{x})$—note that this is still not yet the bolded $\mathbf{z}$—without the imaginary unit as

$$
\begin{aligned}
\boldsymbol{\omega} &\sim p(\boldsymbol{\omega}) \\
b &\sim \mathrm{Uniform}(0, 2\pi) \\
z_{\boldsymbol{\omega}}(\mathbf{x}) &= \sqrt{2}\cos(\boldsymbol{\omega}^\top\mathbf{x} + b).
\end{aligned}
\tag{15}
$$

This works because

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\omega}}[z_{\boldsymbol{\omega}}(\mathbf{x})z_{\boldsymbol{\omega}}(\mathbf{y})] &= \mathbb{E}_{\boldsymbol{\omega}}[\sqrt{2}\cos(\boldsymbol{\omega}^\top\mathbf{x}+b)\sqrt{2}\cos(\boldsymbol{\omega}^\top\mathbf{y}+b)] \\ &\overset{\star}{=} \mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}+\mathbf{y})+2b)] + \mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))] \\ &\overset{\dagger}{=} \mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))].\end{aligned} \tag{16}$$

Step $\star$ is just trigonometry. See A2 for a derivation. Step † uses the fact that since $b \sim \text{Uniform}(0, 2\pi)$, the expectation with respect to b is zero:

$$\mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}+\mathbf{y})+2b)] = \mathbb{E}_{\boldsymbol{\omega}}[\mathbb{E}_b[\cos(\boldsymbol{\omega}^\top(\mathbf{x}+\mathbf{y})+2b) \mid \boldsymbol{\omega}]] = 0 \tag{17}$$

If you are unconvinced, see A3. We are now ready to define the random map $\mathbf{z} : \mathbb{R}^D \mapsto \mathbb{R}^R$ such that (7) holds. Let

$$\mathbf{z}(\mathbf{x}) = \begin{bmatrix} \frac{1}{\sqrt{R}} z_{\boldsymbol{\omega}_1}(\mathbf{x}) \\ \frac{1}{\sqrt{R}} z_{\boldsymbol{\omega}_2}(\mathbf{x}) \\ \vdots \\ \frac{1}{\sqrt{R}} z_{\boldsymbol{\omega}_R}(\mathbf{x}) \end{bmatrix}. \tag{18}$$

and therefore

$$\begin{aligned}\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{y}) &= \frac{1}{R}\sum_{r=1}^{R} z_{\boldsymbol{\omega}_r}(\mathbf{x}) z_{\boldsymbol{\omega}_r}(\mathbf{y}) \\ &= \frac{1}{R}\sum_{r=1}^{R} 2\cos(\boldsymbol{\omega}_r^\top\mathbf{x}+b_r)\cos(\boldsymbol{\omega}_r^\top\mathbf{y}+b_r) \\ &= \frac{1}{R}\sum_{r=1}^{R} \cos(\boldsymbol{\omega}_r^\top(\mathbf{x}-\mathbf{y})) \\ &\approx \mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))] \\ &= k(\mathbf{x},\mathbf{y}).\end{aligned} \tag{19}$$

We now have a simple algorithm to estimate a shift invariant, positive definite kernel. Draw R samples of $\boldsymbol{\omega} \sim p(\boldsymbol{\omega})$ and $b \sim \text{Uniform}(0, 2\pi)$ and then compute $\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{y})$.

An alternative version of random Fourier features that you might see is

$$z_{\boldsymbol{\omega}_r}(\mathbf{x}) = \begin{bmatrix} \cos(\boldsymbol{\omega}_r^\top\mathbf{x}) \\ \sin(\boldsymbol{\omega}_r^\top\mathbf{x}) \end{bmatrix}. \tag{20}$$

See A4 to see why this works and (Sutherland & Schneider, 2015) for a comparative analysis of each approach.

Examples

Gaussian kernel approximation

Before fitting a more complex model, let's first approximate a Gaussian kernel using random Fourier features. Sample R i.i.d. $\boldsymbol{\omega}$ variables from a spherical Gaussian and then compute

$$\mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{y}) = \frac{1}{R}\sum_{r=1}^{R} z_{\boldsymbol{\omega}_r}(\mathbf{x})^\top z_{\boldsymbol{\omega}_r}(\mathbf{y}) = \frac{1}{R}\sum_{r=1}^{R} \cos(\boldsymbol{\omega}_r^\top(\mathbf{x}-\mathbf{y})). \tag{21}$$

for each $(\mathbf{x}, \mathbf{y})$ pair in the data. The resultant $N \times N$ object is the approximate covariance matrix induced by the Gaussian kernel function. Concretely, let $\mathbf{Z}_X$ denote $\mathbf{z}(\cdot)$ applied to all N samples $\mathbf{x}_n$. Thus, $\mathbf{Z}_X$ is $N \times R$ and therefore

$$\mathbf{K}_X \approx \begin{bmatrix} \mathbf{z}(\mathbf{x}_1) \\ \vdots \\ \mathbf{z}(\mathbf{x}_N) \end{bmatrix} \begin{bmatrix} \mathbf{z}(\mathbf{x}_1) & \dots & \mathbf{z}(\mathbf{x}_N) \end{bmatrix} = \mathbf{Z}_X \mathbf{Z}_X^\top \tag{22}$$

because

$$\begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \approx \begin{bmatrix} \mathbf{z}(\mathbf{x}_1)^\top \mathbf{z}(\mathbf{x}_1) & \dots & \mathbf{z}(\mathbf{x}_1)^\top \mathbf{z}(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \mathbf{z}(\mathbf{x}_N)^\top \mathbf{z}(\mathbf{x}_1) & \dots & \mathbf{z}(\mathbf{x}_N)^\top \mathbf{z}(\mathbf{x}_N) \end{bmatrix}. \tag{23}$$

We see in Figure 2 that as R increases, the covariance matrix approximation improves because each cell value uses more Monte Carlo samples to estimate the basis function $\phi(\cdot)$ associated with $k(\cdot, \cdot)$ for the pair of samples associated with that cell. See my GitHub for the code to generate this figure.

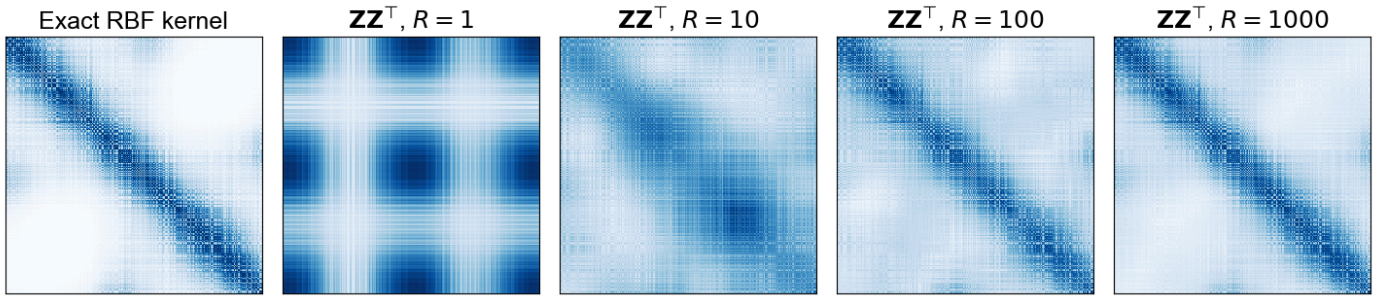

Figure 2. A comparison of a Gaussian kernel on the Scikit-learn S-curve with random Fourier features (left) and an approximated covariance matrix with an increasing number of Monte Carlo samples (right four frames).

Kernel ridge regression

As a more complex example and to see concretely why random Fourier features are efficient, let's look at kernel ridge regression. If needed, see (Welling, 2013) for an introduction to the model. Also, the Scikit-learn docs have a nice tutorial comparing kernel ridge and Gaussian process regression. (8) tells us that $f^*(\mathbf{x})$ is linear in $\mathbf{z}(\mathbf{x})$. Therefore, we just need to convert our input $\mathbf{x}$ into random features and apply linear methods. Concretely, we just want to solve for the coefficients $\boldsymbol{\beta}$ in

$$\hat{\boldsymbol{\beta}} = (\underbrace{\mathbf{Z}_X^\top \mathbf{Z}_X + \lambda \mathbf{I}_R}_{\mathbf{A}})^{-1} \mathbf{Z}_X^\top \mathbf{y}. \tag{24}$$

Above, λ is the ridge regression regularization parameter. See Figure 3 for the results of comparing Gaussian kernel regression with random Fourier feature regression.

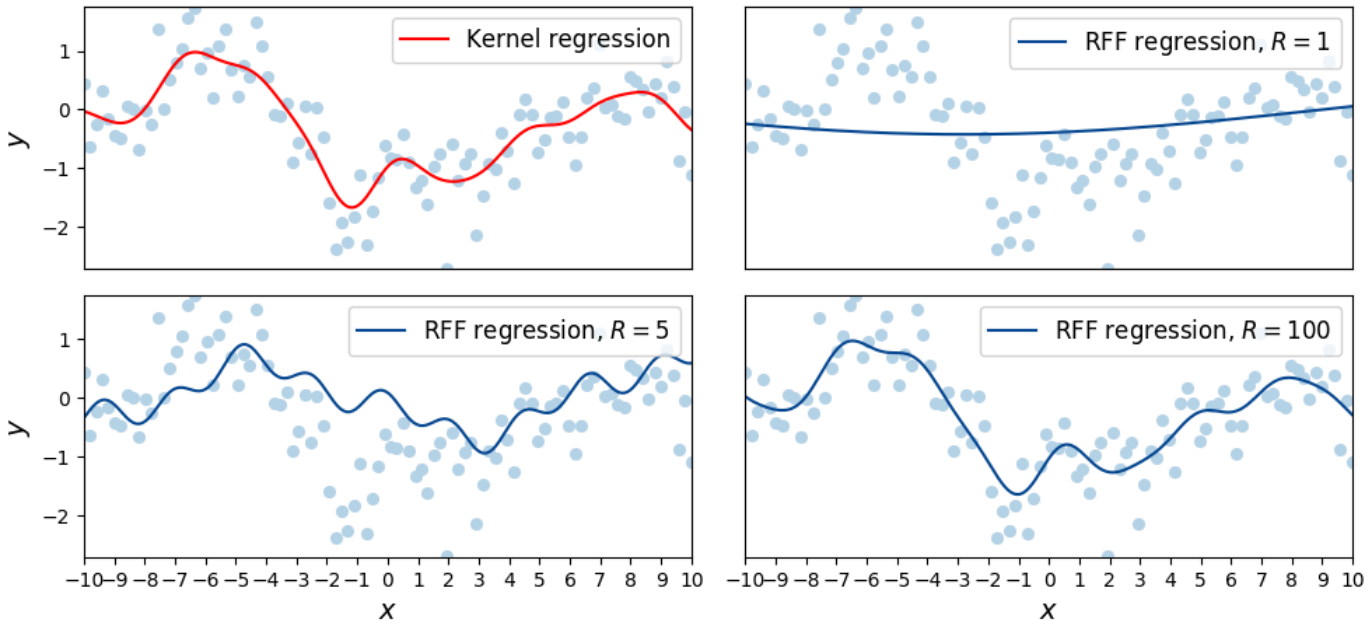

Figure 3. Comparison of Gaussian kernel ridge regression (top left) with ridge regression using random Fourier features (RFF regression) on $N = 100$ data points with $R \in \{1, 5, 100\}$.

With (24) in mind, it is clear why random Fourier features are efficient: inverting $\mathbf{A}$ has time complexity $\mathcal{O}(R^3)$ rather than $\mathcal{O}(N^3)$. If $R \ll N$, then we can have big savings. What is not shown is that even on this small data set, random Fourier feature regression is over an order of magnitude faster than kernel regression with a Gaussian kernel. Since kernel machines scale poorly in N, it is easy to make this multiplier larger by increasing N while keeping R fixed.

Again, see my GitHub for a complete implementation. Note that we need to cache the random variables, $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_R$ and $b_1, b_2, \ldots b_R$, for generating $\mathbf{Z}_X$ so that we have the same transformation when we predict on test data.

Conclusion

Random features for kernel methods is a beautiful, simple, and practical idea; and I see why Rahimi and Recht's paper won the Test of Time Award at NeurIPS 2017. Many theoretical results are impractical or complicated to implement. Many empirical results are not well-understood or brittle. Random features is neither: relatively speaking, it is a simple idea using established mathematics, yet it comes equipped with good theoretical guarantees and good results with practical, easy-to-implement models.

Acknowledgements

I thank Era Choshen for pointing out an error in appendix A3.

Appendix

A1. Gaussian kernel derivation

Let $\boldsymbol{\delta} = \mathbf{x} - \mathbf{y}$:

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{\omega}}[h(\mathbf{x})h(\mathbf{y})^*] \\
&= \mathbb{E}_{\boldsymbol{\omega}}[\exp(i\boldsymbol{\omega}^\top \mathbf{x}) \exp(-i\boldsymbol{\omega}^\top \mathbf{y}))] \\
&= \mathbb{E}_{\boldsymbol{\omega}}[\exp(i\boldsymbol{\omega}^\top \boldsymbol{\delta})] \\
&= \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) \exp(i\boldsymbol{\omega}^\top \boldsymbol{\delta}) \mathrm{d}\boldsymbol{\omega} \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}\boldsymbol{\omega}^\top \boldsymbol{\omega}\Big) \exp(i\boldsymbol{\omega}^\top \boldsymbol{\delta}) \mathrm{d}\boldsymbol{\omega} \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}\boldsymbol{\omega}^\top \boldsymbol{\omega} - i\boldsymbol{\omega}^\top \boldsymbol{\delta}\Big) \mathrm{d}\boldsymbol{\omega} \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}\big(\boldsymbol{\omega}^\top \boldsymbol{\omega} - 2i\boldsymbol{\omega}^\top \boldsymbol{\delta} - \boldsymbol{\delta}^\top \boldsymbol{\delta}\big) - \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\Big) \mathrm{d}\boldsymbol{\omega} \\
&= (2\pi)^{-D/2} \exp\Big(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\Big) \underbrace{\int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}\big(\boldsymbol{\omega} - i\boldsymbol{\delta}\big)^\top \big(\boldsymbol{\omega} - i\boldsymbol{\delta}\big)\Big) \mathrm{d}\boldsymbol{\omega}}_{(2\pi)^{D/2}} \\
&= \exp\Big(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\Big) \\
&= k(\boldsymbol{\delta}).
\end{aligned}
\tag{A1.1}
$$

In other words, $k(\cdot)$ is the Gaussian kernel with $p(\boldsymbol{\omega})$ is a spherical Gaussian.

A2. Trigonometric identity

Recall from trigonometry that

$$\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y). \tag{A2.1}$$

See this Khan Academy video for a proof. Furthermore, note that $\cos(-x) = \cos(x)$ since the cosine function is symmetric about $x = 0$. This is not true of the sine function. Instead, it has odd symmetry: $\sin(-x) = -\sin(x)$. Thus, with a little clever manipulation, we can write

$$\begin{aligned}
&\cos(x+y) + \cos(x-y) \\
&= \cos(x+y) + \cos(x+(-y)) \\
&= [\cos(x)\cos(y) - \sin(x)\sin(y)] + [\cos(x)\cos(-y) - \sin(x)\sin(-y)] \\
&= [\cos(x)\cos(y) - \cancel{\sin(x)\sin(y)}] + [\cos(x)\cos(y) + \cancel{\sin(x)\sin(y)}] \\
&= 2\cos(x)\cos(y).
\end{aligned} \tag{A2.2}$$

A3. Expectation of $\cos(\mathbf{t} + b)$ is zero.

Note that

$$\mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}+\mathbf{y}) + 2b)] = \mathbb{E}_{\boldsymbol{\omega}}[\mathbb{E}_b[\cos(\boldsymbol{\omega}^\top(\mathbf{x}+\mathbf{y}) + 2b) \mid \boldsymbol{\omega}]] \tag{A3.1}$$

holds by the law of total expectation. We claim the inner conditional expectation is zero. To ease notation, let $\mathbf{t} = \boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y})$. Then

$$\begin{aligned}
\mathbb{E}_b[\cos(\mathbf{t}+2b) \mid \boldsymbol{\omega}] &= \int_0^{2\pi} \frac{\cos(\mathbf{t}+2b)}{2\pi} \mathrm{d}b \\
&= \frac{1}{2\pi}\int_0^{2\pi} \cos(\mathbf{t}+2b)\mathrm{d}b \\
&= \frac{1}{2\pi}\Big[\sin(\mathbf{t}+2b)\Big|_0^{2\pi}\Big] \\
&= \frac{1}{2\pi}\Big[\sin(\mathbf{t}) - \sin(\mathbf{t}+4\pi)\Big] \\
&= 0
\end{aligned} \tag{A3.2}$$

The last step holds because $\sin(\mathbf{t}) = \sin(\mathbf{t} \pm 2\pi k)$ for any integer k.

A4. Alternative random Fourier features

Consider this alternative definition of the random map:

$$z_{\boldsymbol{\omega}_r}(\mathbf{x}) = \begin{bmatrix} \cos(\boldsymbol{\omega}_r^\top \mathbf{x}) \\ \sin(\boldsymbol{\omega}_r^\top \mathbf{x}) \end{bmatrix}. \tag{A4.1}$$

Draw $R' = R/2$ samples

$$\boldsymbol{\omega}_r \sim p(\boldsymbol{\omega}). \tag{A4.2}$$

Then

$$\begin{aligned}
\frac{1}{R'}\sum_{r=1}^{R'} z_{\boldsymbol{\omega}_r}(\mathbf{x})^\top z_{\boldsymbol{\omega}_r}(\mathbf{y}) &\equiv \frac{2}{R}\sum_{r=1}^{R/2}\left(\begin{bmatrix}\cos(\boldsymbol{\omega}_r^\top \mathbf{x})\\ \sin(\boldsymbol{\omega}_r^\top \mathbf{x})\end{bmatrix}^\top \begin{bmatrix}\cos(\boldsymbol{\omega}_r^\top \mathbf{y})\\ \sin(\boldsymbol{\omega}_r^\top \mathbf{y})\end{bmatrix}\right) \\
&= \frac{2}{R}\sum_{r=1}^{R/2}\cos(\boldsymbol{\omega}_r^\top \mathbf{x})\cos(\boldsymbol{\omega}_r^\top \mathbf{y}) + \sin(\boldsymbol{\omega}_r^\top \mathbf{x})\sin(\boldsymbol{\omega}_r^\top \mathbf{y}) \\
&\overset{\star}{=} \frac{2}{R}\sum_{r=1}^{R/2}\cos(\boldsymbol{\omega}_r^\top \mathbf{x} - \boldsymbol{\omega}_r^\top \mathbf{y}) \\
&\approx \mathbb{E}_{\boldsymbol{\omega}}[\cos(\boldsymbol{\omega}^\top(\mathbf{x}-\mathbf{y}))] \\
&= k(\mathbf{x}, \mathbf{y}).
\end{aligned} \tag{A4.3}$$

Step $\star$ just applies the product identities from trigonometry:

$$\begin{aligned}
2\sin(x)\sin(y) &= \cos(x-y) - \cancel{\cos(x+y)} \\
2\cos(x)\cos(y) &= \cos(x-y) + \cancel{\cos(x+y)}.
\end{aligned} \tag{A4.4}$$

The right-most terms above cancel in $(\mathrm{A4.3})$, and we get $2\cos(x-y)$.

---

1. Kimeldorf, G., & Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, *33*(1), 82–95.
2. Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. *International Conference on Computational Learning Theory*, 416–426.
3. Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 1177–1184.
4. Rudin, W. (1962). *Fourier analysis on groups* (Vol. 121967). Wiley Online Library.
5. Sutherland, D. J., & Schneider, J. (2015). On the error of random fourier features. *ArXiv Preprint ArXiv:1506.02785*.
6. Welling, M. (2013). Kernel ridge regression. *Max Welling's Classnotes in Machine Learning*, 1–3.