Deep Neural Networks: A Kernel Viewpoint
10716: Advanced Machine Learning
Pradeep Ravikumar

1 Recap: RKHS Kernel Regression

As we have seen, kernel methods (or kernel machines as they are sometimes called) learn functions in a Reproducing Kernel Hilbert Space (RKHS). Recall that an RKHS is specified by a Mercer kernel $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, and loosely, consists of functions of the form $f(x) = \sum_i \alpha_i K(x_i, x)$, where $\{x_i\} \subseteq \mathcal{X}$. More formally, consider the function space:

$$\mathcal{H}_0 = \{f \,:\, f(x) = \sum_{j=1}^{m} \alpha_j K(x_j, x),\ \alpha_j \in \mathbb{R}, x_j \in \mathcal{X}, m \in \mathbb{N}\}.$$

Given two such functions $f = \sum_i \alpha_i K(x_i, x)$ and $g(x) = \sum_j \beta_j K(y_j, x)$, we can define the inner product:

$$\langle f, g\rangle_K = \sum_i \sum_j \alpha_i \beta_j K(x_i, y_j),$$

which then defines a norm $\|f\|_K = \sqrt{\langle f, f\rangle_K}$. The RKHS $\mathcal{H}_K$ corresponding to the kernel K is then the completion of $\mathcal{H}_0$ with respect to the inner product $\|\cdot\|_K$.

(Ridge) RKHS regression, also simply called kernel (ridge) regression in the ML literature (not to be confused with Nadaraya Watson smoothing kernel based non-parametric regression), then consists of solving:

$$\inf_{f \in \mathcal{H}_K} \left\{ \widehat{R}_n(f) + \lambda \|f\|_K^2 \right\},$$

where $\widehat{R}_n(f) = \frac{1}{n}\sum_{i=1}^n \mathcal{L}(f(x_i), y_i)$ is some empirical risk. A typical loss function is the squared loss $\mathcal{L}(f(x_i), y_i) = \frac{1}{2}(f(x_i) - y_i)^2$, in which case the estimate can be shown to be:

$$\widehat{f}_n(x) = K(x, X)(K + \lambda I)^{-1} y,$$

where $K(x, X) = (K(x, x_1), \ldots, K(x, x_n)) \in \mathbb{R}^n$, and $K \in \mathbb{R}^{n \times n}$ is the so-called kernel gram matrix with $K_{ij} = K(x_i, x_j)$, and $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ is the set of responses. In the case where $\lambda = 0$, this yields the estimator: $\widehat{f}_n(x) = K(x, X) K^{-1} y$.

2 Random Features: a two-layer NN from kernels

Let us first review the approach of random features which were developed as a way to speed up kernel machines. As in the previous section, KRR requires inverting the gram matrix

K (or more generally, solving a linear system involving K for any test point x), which is expensive. As we will see we can also view this as distilling a two-layer "neural" network from the kernel machine.

Our starting point is from a spectral decomposition of the kernel. Our Mercer kernel however may not always have a *discrete* spectral decomposition (such an RKHS is called a "separable" RKHS). More generally, the spectral decompositon might have the form:

$$K(x, y) = \int p(w) \phi(w, x) \phi(w, y) dw,$$

where $\{\phi(w, \cdot)\}$ are the eigenfunctions of the kernel, and $\{p(w)\}$ the corresponding "spectral density" (the kernel can be appropriately scaled so that $p(w)$ is a proper density). As before, this also lends itself to an alternative characterization of the RKHS $\mathcal{H}_K$:

$$\mathcal{H}_K = \left\{ f(x) = \int \theta(w) \phi(w, x) dw \mid \int \frac{\theta^2(w)}{p(w)} < \infty \right\}.$$

Example: Shift-invariant kernels As an example of the above decomposition, consider shift-invariant kernels of the form $k(x, y) = k(x - y)$. A classical harmonic analysis result, Bochner's theorem, states that a continuous shift-invariant kernel with domain $\mathbb{R}^d$ is the Fourier transform of a non-negative measure, so that:

$$k(x, y) = k(x - y) = \int_{\mathbb{R}^d} p(w) \exp(j w^T x),$$

where with the kernel appropriately scaled, $p(w)$ is a proper density. When both k and p are real-valued, we can replace the complex exponentials with cosines: $\phi(w, x) = \sqrt{2} \cos(w^T x + b)$, with $b \sim \text{Unif}[0, 2\pi]$.

- Gaussian kernels: $k(x, y) = k(x-y) = \exp(-\|x-y\|^2/2)$, have $p(w) = (2\pi)^{-d} \exp(-\|w\|^2/2)$
- Laplacian kernels: $k(x, y) = k(x - y) = \exp(-\|x - y\|_1)$, have $p(w) = \prod_{j=1}^d \frac{1}{\pi(1+w_j^2)}$
- Cauchy kernels: $k(x, y) = k(x - y) = \prod_{j=1}^d \frac{2}{\pi(1+(x_j-y_j)^2)}$, has $p(w) = \exp(-\|w\|_1)$

So in many such cases the eigenfunctions are a known class of functions such as the Fourier basis for shift-invariant kernels. But when they are not enumerable, how do we extract a finite-dimensional eigenspace? Suppose we draw m random samples $W := \{w_j\}_{j=1}^m$ iid from $p(w)$. This then specifies the following finite-dimensional feature map:

$$\Phi_W(x) = \left(\frac{1}{\sqrt{m}} \phi(w_1, x), \ldots, \frac{1}{\sqrt{m}} \phi(w_m, x) \right),$$

that is indexed by the random draw W. Thus, the feature map $\Phi_W(x)$ itself is random. In contrast to learning a function in the RKHS corresponding to K, one can then simply learn a linear function of these random features, an approach which naturally, is called "random features" (Rahimi & Recht 2008, 2009). But would doing so approximate the RKHS, and would the Euclidean inner product of the random feature map approximate the kernel?

2.1 Approximating the kernel

Consider the empirical kernel $\widehat{K}(x,y) = \frac{1}{m}\sum_{j=1}^{m}\phi(w_j,x)\phi(w_j,y)$, which is simply the Euclidean dot product of the feature maps $\Phi_W(x)$ and $\Phi_W(y)$. Rahimi & Recht (2008) then showed that for any compact set $\mathcal{M}$,

$$\mathbb{P}\left[\sup_{(x,y)\in\mathcal{M}}(\widehat{K}(x,y) - K(x,y)) \geq \epsilon\right] \preceq \frac{\mathrm{diam}^2(\mathcal{M})}{\epsilon^2}\exp(-m\,\epsilon^2).$$

2.2 Approximating the RKHS

Consider the sub-class $\mathcal{H}_C \subset \mathcal{H}_K$ given by:

$$\mathcal{H}_C = \left\{\int \theta(w)\phi(w,x)dw \mid |\theta(w)| < Cp(w)\right\},$$

for some constant $C > 0$. Consider its empirical counterpart:

$$\mathcal{H}_{W;C} = \left\{f(x) = \sum_{j=1}^{m}\theta_j\phi(w_j,x) \mid |\theta_j| \leq C/\sqrt{m}\right\}.$$

Consider the estimator

$$\widehat{f}_W \in \arg\inf_{f\in\mathcal{H}_{W;C}} \widehat{R}_n(f).$$

Rahimi & Recht (2009) then showed that with probability at least $1-\delta$:

$$R(\widehat{f}_W) - \min_{f\in\mathcal{H}_C} R(f) \preceq \left(\frac{1}{\sqrt{m}} + \frac{1}{\sqrt{n}}\right)\sqrt{\log\frac{1}{\delta}}.$$

In particular, they showed that with probability at least $1-\delta$, the approximation error scales as:

$$\inf_{f\in\mathcal{H}_{W;C}} \mathcal{R}(f) - \inf_{f\in\mathcal{H}_C} R(f) \preceq \frac{1}{m},$$

and that the estimation error scales as:

$$\sup_{f \in \mathcal{H}_{W;C}} (\widehat{R}(f) - R(f)) \preceq \frac{1}{n}.$$

Rahimi & Recht (2009) moreover showed that linear combinations of such randomly weighted features $\{\phi(w_j, x)\}$ often outperform approaches that also aim to learn these weights, such as in additive boosting.

2.3 Random Features as a Kernel Machine.

The previous results show that random features i.e. a linear machine on top of a random feature map drawn from the spectral decomposition of the kernel, would *approximate* a kernel machine (i.e. learning a function from the corresponding RKHS) with the kernel K. It is of course also the case that random features are also a kernel machine, just with a "random features kernel" $\widehat{K}(x, y) = \langle \Phi_W(x), \Phi_W(y) \rangle$, and which is an approximation of the kernel K.

2.4 Random Features as a 2 layer network

It can be seen that the RF model has the form:

$$f_{\mathrm{RF}}(x) = \sum_{j=1}^{m} \theta_j \phi(w_j, x).$$

In many settings, for instance, with the Fourier basis for shift-invariant kernels, this further has the form:

$$f_{\mathrm{RF}}(x) = \sum_{j=1}^{m} \theta_j \phi(w_j^T x),$$

so that this can be viewed as a 2 layer NN, with activations ϕ, and m hidden neurons in the 1st layer.

3 Extracting kernels from DNNs

Suppose we have a generic random feature map, $\Phi_W(X)$, that is indexed by a random vector W, but that is not necessarily derived via a spectral decomposition of a Mercer Kernel. Suppose we learn linear functions of this feature map, $f_\theta(x) = \sum_{j=1}^{m} \theta_j \Phi_{W;j}(x)$. We could then consider this as a kernel machine with the random kernel $\widehat{K}(x, y) = \langle \Phi_W(x), \Phi_W(y) \rangle$, which could be viewed as a randomized approximation of the kernel $K(x, y) = \mathbb{E}_W \langle \Phi_W(x), \Phi_W(y) \rangle$.

This broadened perspective of random features can be used to connect DNNs and kernels: if we can approximate a DNN (or any similar complex model) via a linear function of a random feature map, then we could in turn connect it to kernel machines with some specific kernel.

The most natural approach to extract such a random feature map from a DNN is implicit in the common approach of freezing all but the top layer of some pre-trained DNN, and just fitting the top layer. Let $\Phi_W(x)$ denote the mapping from the inputs to the penultimate layer: this is random because the pre-trained DNN weights (in this case, of all but the last layer) depend on random training points, and some random initialization. They thus provide a generic random feature map, that in turn specify the random kernel $\widehat{K}$, with expectation K as detailed above. A simpler instance of this is where we simply randomly initialize all but the top layer, and then just fit the top layer. The corresponding kernel $\widehat{K}$ in that case can be shown to converge to a specific compositional kernel that we will discuss in the next section.

4 Randomly Wired DNNs and Gaussian Processes

We have just seen that just training the top layer of a DNN, while freezing the rest of the layers to some random initialization, or some pre-trained random weights, can be connected to a corresponding kernel machine. But what if we do not train even the top layer, and simply consider a DNN with randomly set weights. Would this still compute something useful? As it turns out, it still serves as a very useful Gaussian Process prior over possible functions (Lee et al. 2017).

Consider an L layer DNN, with n_ℓ units in the ℓ-th layer, for $\ell \in \{0, \ldots, L\}$, where we have $n_0 = d$ features in the input layer. Denote the pre-activation output of the ℓ-th layer as $z^{[\ell]}$, and the output of the ℓ-th layer as $x^{[\ell]} = \phi(z^{[\ell]})$.

Let $x^{[0]} = x$. Then, for $\ell = 1, \ldots, L$, the pre-activation computation at the ℓ-th layer is given as

$$z_i^{[\ell]} = b^{[\ell]} + \sum_{j=1}^{n_{\ell-1}} W_{ij}^{[\ell]} x_j^{[\ell-1]},$$

where $b^{[\ell]}$ is the scalar bias term, and $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$. Denote the set of all parameters by θ, and the final DNN output as $f_\theta(x) = x^{[L]}(x)$.

Consider the random initialization of the parameters as: $W_{ij}^{[\ell]} \sim \mathcal{N}(0, \sigma_w / n_{\ell-1})$, and $b^{[\ell]} \sim \mathcal{N}(0, \sigma_b)$. Since we have that $z_i^{[1]} = b^{[1]} + \sum_{j=1}^{d} W_{ij}^{[1]} x_j$, it follows that $\{z^{[1]}(x)\}$ are Gaussian distributed, and moreover, any collection $(z_i^{[1]}(x_1), \ldots, z_i^{[1]}(x_m))$ is also multivariate Gaussian distributed. It thus follows that $z_i^{[1]}(x)$ is a Gaussian Process with mean $\mu^1(x) = \mathbb{E}[z^{[1]}(x)] =$

0, and covariance:

$$K^{[1]}(x,x') = \mathbb{E}[z_i^{[1]}(x) z_i^{[1]}(x')] = \sigma_b^2 + \sigma_w^2 \frac{x^T x'}{d},$$

where d is the dimensionality of the inputs x.

Let us build this up by induction. Suppose that $z_i^{[\ell-1]}$ is a Gaussian Process. Unfortunately, this does not entail that $x_j^{[\ell-1]}$ is a Gaussian process. But consider the next layer: $z_i^{[\ell]} = b^{[\ell]} + \sum_{j=1}^{n_1} W_{ij}^{[\ell]} x_j^{[\ell-1]}$. In the limit of infinite width, with $n^{\ell-1} \to \infty$, by the Central Limit Theorem, it follows that $z_i^{[\ell]}(x)$ is Gaussian distributed. Moreover, any collection $(z_i^{[\ell]}(x_1), \ldots, z_i^{[2]}(x_m))$ is also multivariate Gaussian distributed. It thus follows that $z_i^{[\ell]}(x)$ is a Gaussian Process with mean $\mu^\ell(x) = \mathbb{E}[z^{[\ell]}(x)] = 0$, and covariance $K^{[\ell]}(x,x') = \sigma_b^2 + \sigma_w^2 \mathbb{E}_{z_i^{[\ell-1]} \sim \mathrm{GP}(0, K^{[\ell-1]})}[\phi(z_i^{[\ell-1]}(x)), \phi(z_i^{[\ell-1]}(x'))]$.

Since the expectation is over some functional of only two zero mean Gaussian variables, we can rewrite as a functional of the covariance entries:

$$K^{[\ell]}(x,x') = \sigma_b^2 + \sigma_w^2 F_\phi(K^{[\ell-1]}(x,x), K^{[\ell-1]}(x,x'), K^{[\ell-1]}(x',x')),$$

for some function $F_\phi(\cdot)$.

Note that in the base case, $K^{[1]}(x,x') = \mathbb{E}[z_i^{[1]}(x) z_i^{[1]}(x')] = \sigma_b^2 + \sigma_w^2 \frac{x^T x'}{d}$, where d is the dimensionality of the inputs x.

It thus follows that at initialization, infinite width DNNs are Gaussian Processes with a specific "deep" or compositional kernel that is as specified by the compositional recurrence above.

Consider the Gaussian Process regression model $y = f(x) + \epsilon$, where $f \sim \mathrm{GP}(0, K^{[L]})$, and $\epsilon \sim N(0, \sigma_\epsilon^2)$. Then, given data $\{(x_i, y_i)\}_{i=1}^n$, as our Bayesian predictor at a test point x, we can simply output the posterior mean:

$$K(x,X)(K(X,X) + \sigma_\epsilon^2 I)^{-1} y,$$

where we have used the shorthand $K(x,X) = (K(x,x_1), \ldots, K(x,x_n))$.

5 Fully trained DNNs, Linearized Models

So far, we have seen that either just training the top layer, freezing the rest, or not training any layer, just using a fully randomly wired DNN, can both be connected to kernel machines. But what if we trained all the DNN layers. Could this still be connected to a kernel machine?

5.1 Kernel Gradient Descent

Suppose we fit a linear model with features $\Phi(\cdot)$, and learn the linear functions $f_\theta(\cdot)$ given training data $\{(x_i, y_i)\}_{i=1}^n$ by solving the objective:

$$\inf_\theta \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f_\theta(x_i), y_i),$$

via gradient descent. The gradient descent path is then given by:

$$\frac{d\theta}{dt} = -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} \Phi(x_i).$$

It thus follows that:

$$\begin{aligned}
\frac{df_\theta(x)}{dt} &= -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} \left\langle \frac{\partial f_\theta(x_i)}{\partial \theta}, \frac{\partial f_\theta(x)}{\partial \theta} \right\rangle \\
&= -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} \langle \Phi(x_i), \Phi(x) \rangle \\
&= -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} K(x_i, x),
\end{aligned}$$

where we use $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$. But of course we could perform the above calculation with any kernel K, and perform what is known as "kernel gradient descent":

$$\frac{df(x)}{dt} = -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} K(x_i, x).$$

This has as its limit point precisely the unregularized RKHS regression estimate with the kernel K. For instance, for the squared loss $\mathcal{L}(f(x), y) = \frac{1}{2}(f(x) - y)^2$, the limit point is precisely $K(x, X)K^{-1}y$. If we can revisit this kernel gradient descent by mapping the gradient descent dynamics when training general non-linear functions (not just linear functions of a given feature map as above) to kernel gradient descent, this can allow us to "extract" a kernel from gradient descent dynamics. This kernel will be referred to as a Neural Tangent Kernel.

Consider training a complex parametric function $f_\theta(x)$ (for instance a DNN) via gradient descent:

$$\frac{d\theta}{dt} = -\eta \frac{1}{n} \sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta} \frac{\partial f_\theta(x_i)}{\partial \theta},$$

so that it follows that:

$$\begin{aligned}\frac{df_\theta(x)}{dt} &= -\eta\,\frac{1}{n}\sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta}\left\langle \frac{\partial f_\theta(x_i)}{\partial \theta}, \frac{\partial f_\theta(x)}{\partial \theta}\right\rangle \\ &= -\eta\,\frac{1}{n}\sum_i \frac{\partial \mathcal{L}(f_\theta(x_i), y_i)}{\partial f_\theta}\widehat{K}_t(x_i, x),\end{aligned}$$

where

$$\widehat{K}_t(x,y) = \left\langle \frac{\partial f_\theta(x)}{\partial \theta}, \frac{\partial f_\theta(y)}{\partial \theta}\right\rangle.$$

When f_θ is a DNN, Jacot et al. (2018) termed the corresponding kernel $\widehat{K}_t$ as the Neural Tangent Kernel at the current iterate θ_t. Jacot et al. (2018) moreover showed the following two results: (a) as the widths of the layers sequentially approach infinity (so-called infinitely wide DNNs), the random NTKs at the initialization $\widehat{K}_0$ converge to a deterministic compositional kernel, which can be derived via a recurrence similar to that of the Gaussian Process kernel above; and (b) uniformly over $t \in [0, T]$ for some finite time horizon $T > 0$, the random NTKs at the iterates θ_t also converge to that same limit determinsitic NTK.

Arora et al. (2019) further refined this analysis providing non-asymptotic deviation bounds. Specifically, they showed that:

Theorem 1 $\forall \epsilon > 0, \delta \in (0,1)$, *suppose that* $\min_{\ell \in [L]} n_\ell \preceq \log(1/\delta)/\epsilon^4$. *Then for any inputs* $x, x' \in \mathbb{R}^d$, *such that* $\|x\| \leq 1, \|x'\| \leq 1$, *with probability at least* $1-\delta$, *there exists a deterministic limit kernel* K_∞ *such that:*

$$|\widehat{K}_t(x,x') - K_\infty(x,x')| \preceq \epsilon.$$

5.2 Linearization

The results above essentially show that in infinite width limit, DNNs are essentially behave similar to the generalized random features machines as in the previous section: linear machines of some random feature maps, which in turn specify some random kernels, which are close to their deterministic expectations. But why do they have such linear behavior?

Consider a complex parametric function f_θ (for instance a DNN). When this function is highly overparameterized, and we take relatively fewer descent steps, then it is likely that each individual parameter moves only slightly away from its initial value, to have the overall function value with its hundreds of millions of parameters to move sufficiently far from its initial value, and in particular to fit the data well. In such cases, where each individual parameter only moves a small amount even with full training, we can approximate the learning of f_θ to simply learning the coefficients of a linearization (Lee et al. 2019):

$$f_w^{\text{LIN}}(x) = f_{\theta_0}(x) + \left\langle w, \frac{\partial f_\theta(x)}{\partial \theta}\Big|_{\theta=\theta_0}\right\rangle.$$

Denote $\phi_{\theta_0}(x) = \frac{\partial f_\theta(x)}{\partial \theta}\big|_{\theta=\theta_0}$. Suppose we initialize θ_0 randomly, for instance via iid standard Gaussians: $[\theta_0]_i \sim N(0,1)$. It can then be seen that training this linear model is essentially random features regression corresponding to the kernel:

$$K_{\text{NTK}}(x, x') = \text{Cov}_{\theta_0}\left(\langle \phi_{\theta_0}(x), \phi_{\theta_0}(x') \rangle\right).$$

In the case of DNNs, this can be seen to be precisely the Neural Tangent Kernel (NTK) at initialization.

5.3 Lazy Training

Chizat et al. (2019) recently argued that such linearization (or what they call "lazy training") is not due to overparameterization per se, but rather due to a particular scaling of the parameters relative to the gradient descent step size. Their high level insight is that if the step size is too small relative to the scale of the parameters, we might naturally expect to be in the "lazy training" or linearized regime where the parameters inividually do not move too far from the initial parameters.

Towards formalizing this, consider the rescaled risk function:

$$R_\alpha(\theta) = \frac{1}{\alpha^2} R(\alpha\, f_\theta),$$

as well as its linearization:

$$R_\alpha^{\text{LIN}}(\theta) = \frac{1}{\alpha^2} R(\alpha\, f_\theta^{\text{LIN}}),$$

where

$$f_\theta^{\text{LIN}}(x) = f_{\theta_0}(x) + \left\langle \theta, \frac{\partial f_\theta(x)}{\partial \theta}\Big|_{\theta=\theta_0} \right\rangle,$$

is the linearization of f_θ around some initial parameter θ_0. Now consider both gradient descent on the full rescaled risk:

$$\frac{d\theta_\alpha}{dt} = -\nabla R_\alpha(\theta),$$

as well as gradient descent on the linearized rescaled risk:

$$\frac{d\theta_\alpha^{\text{LIN}}}{dt} = -\nabla R_\alpha^{\text{LIN}}(\theta).$$

Chizat et al. (2019) then showed that given a finite time horizon $T > 0$:

$$\sup_{t\in[0,T]} \|\theta_\alpha(t) - \theta_0\| = O(1/\alpha) \tag{1}$$

$$\sup_{t\in[0,T]} \|\theta_\alpha(t) - \theta^{\text{LIN}}(t)\| = O(1/\alpha^2) \tag{2}$$

so that the gradient descent iterates remain close to the initial parameter, and the gradient descent path for the linearized function remains close to the path for the original function itself.

6 The representational power of (compositional) kernel methods

The previous sections showed that at least in some over-parameterized or parameter scaling regimes, learning complex functions such as DNNs is essentially similar to kernel machines for some fixed, if a deep or compositionally specified kernel function. But how good are such kernel machines? Do we actually get the flexibility of arbitrary DNNs?

Chizat et al. (2019) empirically showed that lazy training (such as in learning linearized models in overparameterized settings) is much worse empirically than "non-lazy" training where the parameters are no longer in the "lazy training" regimes. Ghorbani et al. (2019) bolstered this with a very powerful set of theoretical results that rigorously analyzed the approximation and generalization errors of such kernel machines, ranging over both random as well as deterministic kernel machines discussed above.

Specifically, they considered the setting where the inputs $x_i \sim \text{Unif}(S^{d-1}(\sqrt{d}))$, where $S^{d-1}(r)$ denotes the sphere with radius r in d dimensions. And the responses $y_i = f^*(x_i)$, for some arbitrary $f^* : S^{d-1}(\sqrt{d}) \mapsto \mathbb{R}$.

Consider RF machines:

$$f_{\text{RF}}(x) = \sum_{j=1}^{N} a_j \sigma(w_j^T x),$$

where $w_j \sim p(w)$, for $j \in [N]$, and the spectral density p of some fixed kernel. They also consider the NTK kernel of a 2 layer NN. Such a 2 layer NN is given by:

$$f_{\text{NN}}(x) = \sum_{j=1}^{N} a_j \sigma(w_j^T x).$$

Suppose the weights are initialized given the spectral density p. It can be seen that the NTK feature map (obtained by gradients with respect to a and w, is the concatenation of the RF feature map and the NTK-specific features:

$$f_{\text{NTK}}(x) = \sum_{j=1}^{N} \theta_j \sigma(w_j^T x) + \sum_{j=1}^{N} ({\theta'}_j^T x) \sigma'(w_j^T x).$$

For the RF machines, let N be the dimensionality of finite-dimensional random feature maps. They then showed that if $d^{\ell+\delta} \leq N \leq d^{\ell+1-\delta}$, then the RF machines have the same approximation error as degree ℓ polynomial in x, while the NTK kernel machines have the same approximation error as degree $\ell + 1$ polynomial in x.

But what if we use the full kernel (rather than just N random features), which might have zero or negligible approximation error. But they might nonetheless incur estimation error given finite n samples. They considered this for general deterministic kernels that are rotationally invariant over the sphere $S^{d-1}(\sqrt{d})$. In such a setting, when $d^{\ell+\delta} \leq n \leq d^{\ell+1-\delta}$, the generalization error of these kernel machines scales as the approximation error of a degree ℓ polynomial in x.

They also coupled this with a very simple separation result to show that this does not entail that DNNs itself have large approximation or generalization error. Specifically, consider the case where the target function $f^* = \sigma(w^* \cdot x)$ is a single neuron. In such a case, under the distributional assumptions on the training inputs above, it can be shown that gradient descent over the empirical mean squared error recovers the true parameter w^*. But using say the NTK kernel machine with $d^{\ell+\delta} \leq N \leq d^{\ell+1-\delta}$ would have approximation error of linear regression over all monomials of degree $\ell + 1$ in x, which could be large if σ cannot be approximated well by polynomials of degree atmost ℓ.

Their results thus provide a neat separation result between general NNs on the one hand, and fixed kernel machines (the full kernel machine i.e. KRR, as well as RF and NTK machines) on the other hand, which are more akin to polynomial regression. Together with the lazy training result, this suggests that fixed kernels that do not use the data distribution might not be a good explanation for DNN performance. In later lectures, we will thus revisit this question of connecting DNNs to (surprisingly simple) kernels, but ones that depend on the data distribution.

References

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R. & Wang, R. (2019), On exact computation with an infinitely wide neural net, *in* 'Advances in Neural Information Processing Systems', pp. 8139–8148.

Chizat, L., Oyallon, E. & Bach, F. (2019), On lazy training in differentiable programming, *in* 'Advances in Neural Information Processing Systems', pp. 2933–2943.

Ghorbani, B., Mei, S., Misiakiewicz, T. & Montanari, A. (2019), 'Linearized two-layers neural networks in high dimension', *arXiv preprint arXiv:1904.12191* .

Jacot, A., Gabriel, F. & Hongler, C. (2018), Neural tangent kernel: Convergence and generalization in neural networks, *in* 'Advances in neural information processing systems', pp. 8571–8580.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. & Sohl-Dickstein, J. (2017), 'Deep neural networks as gaussian processes', *arXiv preprint arXiv:1711.00165* .

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J. & Pennington, J. (2019), Wide neural networks of any depth evolve as linear models under gradient descent, *in* 'Advances in neural information processing systems', pp. 8570–8581.

Rahimi, A. & Recht, B. (2008), Random features for large-scale kernel machines, *in* 'Advances in neural information processing systems', pp. 1177–1184.

Rahimi, A. & Recht, B. (2009), Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning, *in* 'Advances in neural information processing systems', pp. 1313–1320.