Deep Learning

Lecture 06

Random Feature Regression and Neural Tangent Kernel

Aurelien Lucchi

Fall 2024

Section 1

Kernels: a brief introduction/recap

Typical ML Problem

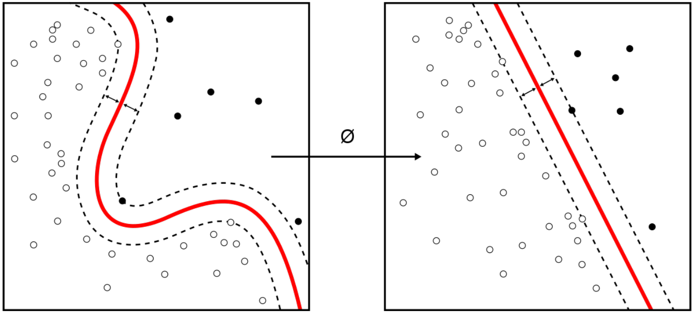
Given some training data $(\mathbf{x}_i, y_i)_{i=1}^n$ where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$, and y_i are labels, e.g. $y_i \in \mathbb{R}$ for regression or $y_i \in \{-1, +1\}$ for classification.

Simplest model: Use a linear model to classify/regress the data.

Question: What do we do if the data is not linearly separable?

Motivation Kernel

Solution: Use a **feature map** $\phi : \mathbb{R}^d \mapsto \mathcal{X}$ to map the data into a (typically high-dimensional) vector space $\mathcal{X}$ where linear relations exist among the data.



Motivation Kernel

Problem: The new feature space given by $\phi(\mathbf{x})$ might be very large (potentially infinite).

$\Longrightarrow$ Instead of directly finding a decision function f in this new space, we will use a similarity metric between the datapoints.

$\Longrightarrow$ **This is where kernels come into play...**

Kernel: Given two datapoints $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ and a map $\phi : \mathcal{X} \to \mathbb{R}^p$, a kernel function K is defined as:

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle.$$

Example: Polynomial kernel in 2-dimension with **feature map** $\phi : \mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.

Kernel Learning (I)

Predictive Function: Given a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^p$, we study functions of the form

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \phi(\mathbf{x})$$

where $\boldsymbol{\theta} \in \mathbb{R}^p$ are the learnable parameters of the model.

Loss: We aim to choose $\boldsymbol{\theta} \in \mathbb{R}^p$ such that we minimize the least-squares loss, i.e.

$$L(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i\right)^2 = \frac{1}{2} \|\boldsymbol{\phi}\theta - \mathbf{y}\|_2^2$$

Kernel Learning (II)

Notations:

- $\boldsymbol{\phi} \in \mathbb{R}^{n \times p}$ with $\phi_i = \phi(\mathbf{x}_i)$
- $\boldsymbol{K}_{\mathbf{x}} \in \mathbb{R}^{n}$ with $(\boldsymbol{K}_{\mathbf{x}})_i = K(\mathbf{x}, \mathbf{x}_i) = \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$
- $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ with $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$

Closed-form: It turns out that we can find the optimal function as

$$f_{\boldsymbol{\theta}^*}(\mathbf{x}) = \phi(\mathbf{x})\boldsymbol{\phi}^T \left(\boldsymbol{\phi}\boldsymbol{\phi}^T\right)^{-1} \boldsymbol{y} = \boldsymbol{K}_{\mathbf{x}}\boldsymbol{K}^{-1}\boldsymbol{y},$$

where we assume for convenience that $\boldsymbol{K}$ is invertible.

Kernel Trick: Notice that we can write $f_{\boldsymbol{\theta}^*}$ only in terms of K, we never need to compute the high-dimensional representation $\phi(\mathbf{x})$!

Connection to Gradient Learning

Gradient Flow:

$$\frac{d}{dt}\boldsymbol{\theta}_t = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$$

In essence, this is gradient descent with an infinitesimal learning rate.

Training Dynamics: We can write down the evolution of the function $f_{\boldsymbol{\theta}_t}$ at any timestep $t \geq 0$ (solving above ODE):

$$f_{\boldsymbol{\theta}_t}(\mathbf{x}) = \boldsymbol{K}_{\mathbf{x}}\boldsymbol{K}^{-1}\left(\mathbf{1}_n - e^{-\boldsymbol{K}t}\right)\boldsymbol{y}$$

where $e^{\boldsymbol{A}}$ is the matrix exponential. Notice that $f_{\boldsymbol{\theta}_\infty} = f_{\boldsymbol{\theta}^*}$.

Section 2

Random Feature Regression

Notations

Notation	Space	Definition
n	$\mathbb{N}$	Number of training datapoints
$\mathbf{x}$	$\mathbb{R}^d$	Datapoint
θ	$\mathbb{R}^p$	Parameters
$\phi(\mathbf{x})$	$\mathbb{R}^d \to \mathbb{R}^p$	Feature map
$\mathbf{X}$	$\mathbb{R}^{n\times d}$	Data matrix
$\mathbf{y}$	$\mathbb{R}^n$	Vector of labels
$L(\theta)$	$\mathbb{R}^p \to \mathbb{R}$	Loss function
K	$\mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$	Kernel

Model

Random feature model: We now study a **specific feature map** given by $\phi_i(\mathbf{x}) = \varphi(\mathbf{w}_i \cdot \mathbf{x})$ where

- $\mathbf{w}_i \in \mathbb{R}^d$, $i = 1, \dots, p$ are **independent random variables**
- $\varphi : \mathbb{R} \to \mathbb{R}$ is some non-linearity function

Example: one hidden layer neural network where only train the top layer.

We then combine the features linearly to construct a model:

$$\boxed{f(\mathbf{x}, \theta) = \frac{1}{\sqrt{p}} \theta^\top \phi(\mathbf{x}),} \tag{model}$$

where $\theta \in \mathbb{R}^p$ are all the model parameters.

The $\frac{1}{\sqrt{p}}$ scaling is used to make sure that the sum does not explode when we take the limit $p \to \infty$.

Role of the Kernel

Next, we will see that the **kernel** K does two things:

1. It determines the evolution of $f(\mathbf{x}, \theta_t)$ (Theorem 1)
2. Under Gaussian initialization, it determines the distribution of $f(\mathbf{x}, \theta_0)$ (Theorem 2).

Evolution of f

Let $f := f^k$ be the output a neural network with k layers.

Theorem 1 (Evolution of f)

For $\mathbf{x} \in \mathbb{R}^d$*:*

$$\frac{d}{dt} f(\mathbf{x}, \theta_t) = K(\mathbf{x}, \mathbf{X})(\mathbf{y} - f(\mathbf{X}, \theta_t)),$$

where $K(\mathbf{x}, \mathbf{X}) \in \mathbb{R}^{1 \times n}$ *and* $(\mathbf{y} - f(\mathbf{X}, \theta_t)) \in \mathbb{R}^{n \times 1}$*.*

The solution of this differential equation is

$$f(\mathbf{x}, \theta_\infty) - f(\mathbf{x}, \theta_0) = K(\mathbf{x}, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1}(\mathbf{y} - f(\mathbf{X}, \theta_0)),$$

where $K(\mathbf{X}, \mathbf{X}) = [K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j} \in \mathbb{R}^{n \times n}$*.*

Proof - Evolution of f

Proof - Evolution of f

Proof - Evolution of f

Definition - GP

Gaussian Process (GP): A GP is a (potentially infinite) collection of random variables (RV) $\{Z_t\}_{t \in S}$ defined over a set S such that the joint distribution of every finite subset of RVs is multivariate Gaussian, i.e.

$$\forall n \in \mathbb{N}, \forall t_1, \ldots t_n \in S, (Z_{t_1}, \ldots, Z_{t_n}) \text{ is Gaussian}$$

Notation: Will typically write $f \sim \mathbf{GP}(\mu, \Sigma)$ to say that f is a GP with mean μ and covariance Σ.

Samples from a GP

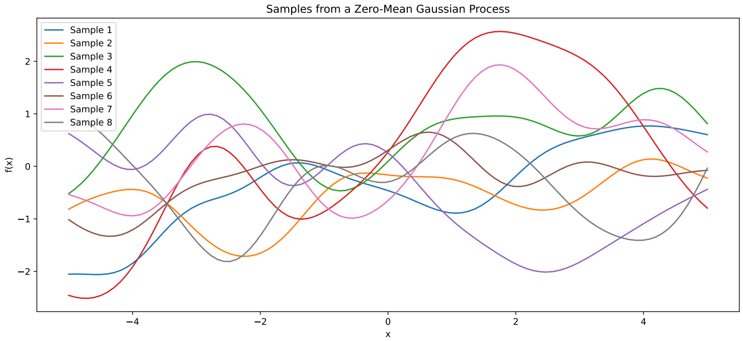


Figure: Samples drawn from a Gaussian process prior $\mathcal{GP}(0, K_{\mathsf{RBF}})$.

Distribution of f

Assume θ_0 is chosen so that all parameters are independent Gaussians with mean 0 and variance 1.

$\rightsquigarrow$ This random initialization of θ induces a distribution over $f(\mathbf{x};\theta)$ whose statistics we will analyze, both at initialization and throughout training (gradient descent on a specified dataset).

Ensemble of neural networks One can think of each draw of θ_0 as a different neural network: each network in the ensemble represents a different realization of the parameter perturbations.

Distribution of f

Consider a neural network $f(\mathbf{x}, \theta)$ trained on inputs drawn from the unit circle.

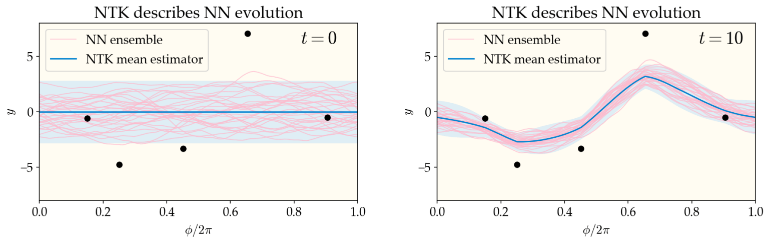


Figure: Source: `https://en.wikipedia.org/wiki/Neural_tangent_kernel`

- At initialization, an ensemble of wide neural networks is a zero-mean Gaussian process
- During training (gradient descent on mean-square error), the ensemble evolves according to the kernel

Initial value of f

At initialization, $f(\mathbf{x}, \theta_0) \sim \textbf{GP}(0, K(\mathbf{x}, \mathbf{x}'))$.

Theorem 2 (Initial value)

Assume θ_0 is chosen so that all parameters are independent Gaussians with mean 0 and variance 1. Then $f(\mathbf{x}, \theta_0)$ is also Gaussian with

$$\mathbb{E}[f(\mathbf{x}, \theta_0)] = 0, \quad \mathrm{var}[f(\mathbf{x}, \theta_0)] = K(\mathbf{x}, \mathbf{x})$$

Also, Cov$[f(\mathbf{x}, \theta_0), f(\mathbf{x}', \theta_0)] = K(\mathbf{x}, \mathbf{x}')$.

Proof idea.
Since $f(\mathbf{x}, \theta_0) = \theta_0^\top \phi(\mathbf{x})$, it's a linear combination of Gaussians, which is Gaussian. Indeed if $\mathbf{x} \sim \mathcal{N}(\mu_\mathbf{x}, \Sigma_\mathbf{x})$, and $\mathbf{y} = a + \mathbf{B}\mathbf{x}$, where $\mathbf{B}$ is a matrix, then $\mathbf{y} \sim \mathcal{N}(a + \mathbf{B}\mu_x, \mathbf{B}\Sigma_\mathbf{x}\mathbf{B}^\top)$. □

Value at $t \to \infty$

Finally, we look at what happens at time $t \to \infty$. Recall that $K(\mathbf{x}, \mathbf{X}) \in \mathbb{R}^{1\times n}, K(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n\times n}, \mathbf{y} \in \mathbb{R}^{n\times 1}$. We will see that: $f(\mathbf{x}, \theta_\infty) \sim$ **GP**$(m(x), \Sigma(\mathbf{x}, \mathbf{x}'))$ where $\Sigma(\mathbf{x}, \mathbf{x}')$ depends on $K(\mathbf{x}, \mathbf{x}')$.

Theorem 3 (Value at $t \to \infty$)

Assuming Gaussian initialization, at $t \to \infty$, we get

$$\mathbb{E}[f(\mathbf{x}, \theta_\infty)] = K(\mathbf{x}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y}$$

and

$$\mathrm{var}[f(\mathbf{x}, \theta_\infty)] = K(\mathbf{x}, \mathbf{x}) - K(\mathbf{x}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{x}, \mathbf{X})$$

Proof: Value at $t \to \infty$

i) From Theorem 2, we have $\begin{pmatrix} f(\mathbf{x}, \theta_0) \\ f(\mathbf{X}, \theta_0) \end{pmatrix}$ is a GP with mean 0 and covariance

$$\begin{pmatrix} K(\mathbf{x}, \mathbf{x}) & K(\mathbf{x}, \mathbf{X}) \\ K(\mathbf{X}, \mathbf{x}) & K(\mathbf{X}, \mathbf{X}) \end{pmatrix}$$

ii) From Theorem 1, we also have

$$\begin{pmatrix} f(\mathbf{x}, \theta_\infty) \\ f(\mathbf{X}, \theta_\infty) \end{pmatrix} = \begin{pmatrix} 1_{1\times 1} & -K(\mathbf{x}, \mathbf{X})K^{-1}(\mathbf{X}, \mathbf{X}) \\ 0_{n\times 1} & 0_{n\times n} \end{pmatrix} \begin{pmatrix} f(\mathbf{x}, \theta_0) \\ f(\mathbf{X}, \theta_0) \end{pmatrix} + \begin{pmatrix} K(\mathbf{x}, \mathbf{X})K^{-1}(\mathbf{X}, \mathbf{X})\mathbf{y} \\ 0_{n\times 1} \end{pmatrix}$$

The result is a Gaussian that is multiplied by a matrix and adding a shift $\implies$ This is a Gaussian.

Section 3

Wide neural networks

Definition Neural Network

First layer: Define the first layer as

$$f^1(\mathbf{x}, \theta) = \frac{\sigma_w}{\sqrt{n_1}} \mathbf{W}^1 \mathbf{x} + \sigma_b \mathbf{b}^1,$$

where $\mathbf{W}^1 \in \mathbb{R}^{n_1 \times d}$ is the weight matrix and $\mathbf{b}^1 \in R^{n_1}$ the bias of the first layer. The parameters (σ_w, σ_b) are the standard deviation of the weights and biases.

The division by $\frac{1}{\sqrt{n_1}}$ is to ensure that we get proper convergence when we take the limit of infinitely wide networks $n_1 \to \infty$.

Post-activation: defined as $h^1(\mathbf{x}, \theta) = \phi(f^1(\mathbf{x}, \theta))$ where ϕ is some entrywize non-linear activation function.

Definition Neural Network

l-th layer: Similarly, for the l-th layer,

$$f^l(\mathbf{x}, \theta) = \frac{\sigma_w}{\sqrt{n_l}} \mathbf{W}^l h^{l-1}(\mathbf{x}, \theta) + \sigma_b \mathbf{b}^l,$$

where $\mathbf{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix and $\mathbf{b}^l \in R^{n_l}$ the bias of the l-th layer.

Note: θ contains all the weight matrices $\mathbf{W}^l$ and biases $\mathbf{b}^l$.

Network output: The output of a network with L layers is $f(\mathbf{x}, \theta) = f^L(\mathbf{x}, \theta)$.

Initialization: The parameter θ contains all the weight matrices $\mathbf{W}^l$ and biases $\mathbf{b}^l$. At iteration 0, θ_0 is random.

General idea

We will see that very wide neural networks are "close" (in terms of Taylor series approximation) to a random feature regression model.

Consider the **linearization** of $f(\mathbf{x}, \theta)$:

$$f^{lin}(\mathbf{x}, \theta) = f(\mathbf{x}, \theta_0) + \underbrace{\nabla_\theta \mathbf{f}(\mathbf{x}, \theta_\mathbf{0})^\top}_{\text{Features}} \underbrace{(\theta - \theta_0)}_{\text{Weights}}$$

$\Longrightarrow f^{lin}(\mathbf{x}, \theta)$ is a feature regression model where the features are given by $\phi(\mathbf{x}) = \nabla_\theta \mathbf{f}(\mathbf{x}, \theta_\mathbf{0})$.

Lee et al. (2019) showed that

$$\|f^{lin}(\mathbf{x}, \theta) - f(\mathbf{x}, \theta_t)\| = \mathcal{O}\left(\frac{1}{\sqrt{p}}\right)$$

where p is the number of parameters.

Kernels: Feature regression vs wide network

	Feature regression	**Wide network**
Initialization	$K(\mathbf{x}, \mathbf{x}')$	NNGP $\Sigma^{L+1}(\mathbf{x}, \mathbf{x}')$
Training	$K(\mathbf{x}, \mathbf{x}') = \frac{1}{p}\phi(\mathbf{x})^\top \phi(\mathbf{x}')$	NTK $\Theta^{L+1}(\mathbf{x}, \mathbf{x}')$

Comments:

- The formulas for Σ and Θ are **recursive** (details later)
- The NTK depends on the NNGP

At initialization

Theorem 4 (Distribution at initialization)

For any point $\mathbf{x}$, $f^l(\mathbf{x}, \theta_0)$ *is Gaussian with*

$$\mathbb{E}[f^l(\mathbf{x}, \theta_0)] = 0 \qquad \mathrm{var}[f^l(\mathbf{x}, \theta_0)] = \Sigma^l(\mathbf{x}, \mathbf{x}),$$

where $\Sigma^l(\mathbf{x}, \mathbf{x}')$ *is defined recursively as*

$$\Sigma^1(\mathbf{x}, \mathbf{x}') = \sigma_w^2 \langle \mathbf{x}, \mathbf{x}' \rangle + \sigma_b^2, \quad \Sigma^l(\mathbf{x}, \mathbf{x}') = \sigma_w^2 \mathbb{E}_{\mathbf{z}, \mathbf{z}'}[\phi(\mathbf{z})^\top \phi(\mathbf{z}')] + \sigma_b^2$$

with

$$\begin{pmatrix} \mathbf{z} \\ \mathbf{z}' \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \Sigma^{l-1}(\mathbf{x}, \mathbf{x}) & \Sigma^{l-1}(\mathbf{x}, \mathbf{x}') \\ \Sigma^{l-1}(\mathbf{x}', \mathbf{x}) & \Sigma^{l-1}(\mathbf{x}', \mathbf{x}') \end{pmatrix}\right)$$

Proof idea.
By induction: show that if one layer is a Gaussian process, then the next layer is also a Gaussian process. □

Definition NTK Kernel

While the NNGP kernel controlled the network at initialization, the training dynamics depends on the NTK kernel $\Theta^l(\mathbf{x}, \mathbf{x}')$ which is defined as follows:

$$\Theta^l(\mathbf{x}, \mathbf{x}') = \Theta^{l-1}(\mathbf{x}, \mathbf{x}')\dot{\Sigma}(\mathbf{x}, \mathbf{x}') + \Sigma(\mathbf{x}, \mathbf{x}'),$$

where

$$\Sigma(\mathbf{x}, \mathbf{x}') = \sigma_w^2 \mathbb{E}[\phi(\mathbf{z})\phi(\mathbf{z}')] + \sigma_b^2$$

$$\dot{\Sigma}(\mathbf{x}, \mathbf{x}') = \sigma_w^2 \mathbb{E}[\dot{\phi}(\mathbf{z})\dot{\phi}(\mathbf{z}')]$$

Training dynamics

Theorem 5 (NTK Kernel)

There is a kernel $\Theta^l(\mathbf{x}, \mathbf{x}')$ *such that, for the layer widths* $n_1, \ldots n_L \to \infty$,

$$\mathbb{E}[\nabla_\theta f^l(\mathbf{x}, \theta_0)^\top \nabla_\theta f^l(\mathbf{x}', \theta_0)] \to \Theta^l(\mathbf{x}, \mathbf{x}')$$

and

$$\frac{d}{dt} f(\mathbf{x}, \theta_t) = -\Theta^{L+1}(\mathbf{x}, \mathbf{X})(f(\mathbf{X}, \theta_t) - \mathbf{y})$$

Proof idea.

- Similar to the proof for the feature regression model.
- As the ensemble size becomes large, the CLT suggests that the ensemble's behavior will tend to follow a normal distribution.

□

Training dynamics

The theorem says that the NTK kernel **controls the dynamics of the neural network**, the same way the kernel K controls the dynamics of the feature regression model.

Summary

- The random initialization of θ induces a distribution over $f(\mathbf{x};\theta)$
- At initialization (before training), the neural network ensemble is a zero-mean Gaussian process (GP) whose covariance $\mathbb{E}_\theta[f(\mathbf{x};\theta)f(\mathbf{x}';\theta)] = \Sigma(\mathbf{x},\mathbf{x}')$, where the GP covariance $\Sigma(\mathbf{x},\mathbf{x}')$ depends on the kernel $K(\mathbf{x},\mathbf{x}')$
- During training, the neural network is linearized, i.e., its parameter dependence can be captured by its first-order Taylor expansion:

$$f^{lin}(\mathbf{x},\theta) = f(\mathbf{x},\theta_0) + \underbrace{\nabla_\theta \mathbf{f}(\mathbf{x},\theta_\mathbf{0})^\top}_{\text{Features}} \underbrace{(\theta - \theta_0)}_{\text{Weights}}$$

  - This linearization is only valid when θ is close to θ_0 (the distance between θ and θ_0 scales inversely with the number of parameters)

Limitation of Kernels

The jury is still out ...

From Ghorbani et al. (2020), *When Do Neural Networks Outperform Kernel Methods?*:

- Some empirical studies on various datasets showed that "networks can be replaced by suitable kernels with limited drop in performances"
- Others show larger gaps empirically
- "Theoretical analysis provided a number of separation examples, i.e. target functions f^* that can be represented and possibly efficiently learnt using neural networks, but not in the corresponding RKHS"

Resources

Lectures Notes on "Random Feature Regression and Wide Neural Networks" from Mihai Nica