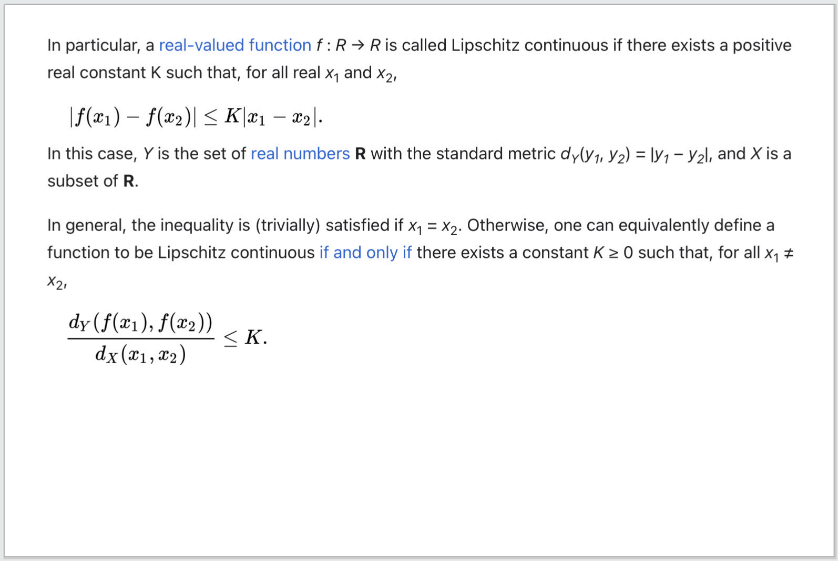
Workspace 'Workspace' in 'Some Math behind Neural Tangent Kernel Lil Log'

Page 1 (row 1, column 1)



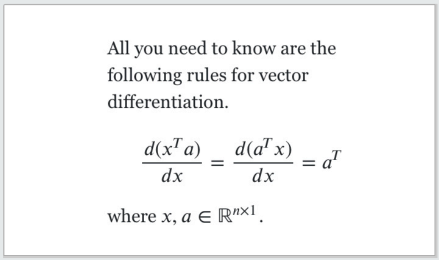
In particular, a real-valued function $f : R \to R$ is called Lipschitz continuous if there exists a positive real constant K such that, for all real x_1 and x_2,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|.$$

In this case, Y is the set of real numbers **R** with the standard metric $d_Y(y_1, y_2) = |y_1 - y_2|$, and X is a subset of **R**.

In general, the inequality is (trivially) satisfied if $x_1 = x_2$. Otherwise, one can equivalently define a function to be Lipschitz continuous if and only if there exists a constant $K \geq 0$ such that, for all $x_1 \neq x_2$,

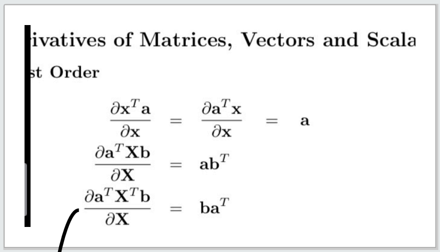
$$\frac{d_Y(f(x_1), f(x_2))}{d_X(x_1, x_2)} \leq K.$$



All you need to know are the following rules for vector differentiation.

$$\frac{d(x^T a)}{dx} = \frac{d(a^T x)}{dx} = a^T$$

where $x, a \in \mathbb{R}^{n \times 1}$.



ivatives of Matrices, Vectors and Scala

st Order

$$\frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T$$

In our case, a is Identity

(Check 3rd) b is x.

x is W.

$$\frac{\partial W^T x}{\partial W} = x$$

Source ↓

https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Lil'Log

Posts Archive Search Tags FAQ emojisearch.app

Some Math behind Neural Tangent Kernel

September 8, 2022 · 17 min · Lilian Weng

▼ Table of Contents

Neural networks are well known to be over-parameterized and can often easily fit data with near-zero training loss with decent generalization performance on test dataset. Although all these parameters are initialized at random, the optimization process can consistently lead to similarly good outcomes. And this is true even when the number of model parameters exceeds the number of training data points.

Neural tangent kernel (NTK) (Jacot et al. 2018) is a kernel to explain the evolution of neural networks during training via gradient descent. It leads to great insights into why neural networks with enough width can consistently converge to a global minimum when trained to minimize an empirical loss. In the post, we will do a deep dive into the motivation and definition of NTK, as well as the proof of a deterministic convergence at different initializations of neural networks with infinite width by characterizing NTK in such a setting.

> 🤓 Different from my previous posts, this one mainly focuses on a small number of core

> papers, less on the breadth of the literature review in the field. There are many interesting works after NTK, with modification or expansion of the theory for understanding the learning dynamics of NNs, but they won't be covered here. The goal is to show all the math behind NTK in a clear and easy-to-follow format, so the post is quite math-intensive. If you notice any mistakes, please let me know and I will be happy to correct them quickly. Thanks in advance!

Basics

This section contains reviews of several very basic concepts which are core to understanding of neural tangent kernel. Feel free to skip.

Vector-to-vector Derivative

Given an input vector $\mathbf{x} \in \mathbb{R}^n$ (as a column vector) and a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the derivative of f with respective to $\mathbf{x}$ is a $m \times n$ matrix, also known as *Jacobian matrix*:

$$\boxed{J = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}}$$

Throughout the post, I use integer subscript(s) to refer to a single entry out of a vector or matrix value; i.e. x_i indicates the i-th value in the vector $\mathbf{x}$ and $f_i(.)$ is the i-th entry in the output of the function.

The gradient of a vector with respect to a vector is defined as $\nabla_\mathbf{x} f = J^\top \in \mathbb{R}^{n \times m}$ and this formation is also valid when $m = 1$ (i.e., scalar output).

Differential Equations

Differential equations describe the relationship between one or multiple functions and their derivatives. There are two main types of differential equations.

- (1) *ODE (Ordinary differential equation)* contains only an unknown function of one random variable. ODEs are the main form of differential equations used in this post. A general form of ODE looks like $(x, y, \frac{dy}{dx}, \dots, \frac{d^n y}{dx^n}) = 0$.
- (2) *PDE (Partial differential equation)* contains unknown multivariable functions and their partial derivatives.

Let's review the simplest case of differential equations and its solution. *Separation of variables* (Fourier method) can be used when all the terms containing one variable can be moved to one side, while the other terms are all moved to the other side. For example, Fourier Method (Separation of Variables)

$$\text{Given } a \text{ is a constant scalar:} \quad \frac{dy}{dx} = ay$$

$$\text{Move same variables to the same side:} \quad \frac{dy}{y} = adx$$

$$\text{Put integral on both sides:} \quad \int \frac{dy}{y} = \int adx$$

$$\ln(y) = ax + C'$$

$$\text{Finally} \quad y = e^{ax+C'} = Ce^{ax}$$

Central Limit Theorem

Given a collection of i.i.d. random variables, $x_1, \dots, x_N$ with mean μ and variance σ^2, the *Central Limit Theorem (CTL)* states that the expectation would be Gaussian distributed when N becomes really large.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \sim \mathcal{N}(\mu, \frac{\sigma^2}{n}) \quad \text{when } N \to \infty$$

CTL can also apply to multidimensional vectors, and then instead of a single scale σ^2 we need to compute the covariance matrix of random variable Σ. → Covariance Matrix used in case of multi-dimensional

Taylor Expansion

The *Taylor expansion* is to express a function as an infinite sum of components, each represented in terms of this function's derivatives. The Tayler expansion of a function $f(x)$ at $x = a$ can be written as:

$$\boxed{f(x) = f(a) + \sum_{k=1}^{\infty} \frac{1}{k!}(x-a)^k \nabla_x^k f(x)|_{x=a}}$$

where ∇^k denotes the k-th derivative.

The first-order Taylor expansion is often used as a linear approximation of the function value:

$$f(x) \approx f(a) + (x-a)\nabla_x f(x)|_{x=a}$$

→ Good approximatio

Kernel & Kernel Methods

A *kernel* is essentially a similarity function between two data points, $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. It describes how sensitive the prediction for one data sample is to the prediction for the other; or in other words, how similar two data points are. The kernel should be symmetric, $K(x, x') = K(x', x)$. → Symmetric

Depending on the problem structure, some kernels can be decomposed into two feature maps, one corresponding to one data point, and the kernel value is an inner product of these two features: $K(x, x') = \langle \varphi(x), \varphi(x') \rangle$.

→ Inner Product of two feature maps.

Kernel methods are a type of non-parametric, instance-based machine learning algorithms. Assuming we have known all the labels of training samples $\{x^{(i)}, y^{(i)}\}$, the label for a new input x is predicted by a weighted sum $\sum_i K(x^{(i)}, x) y^{(i)}$.

$\sum_i K(x^{(i)}, x) y^{(i)}$

Gaussian Processes

Gaussian process (GP) is a non-parametric method by modeling a multivariate Gaussian probability distribution over a collection of random variables. GP assumes a prior over functions and then updates the posterior over functions based on what data points are observed.

Given a collection of data points $\{x^{(1)}, \dots, x^{(N)}\}$, GP assumes that they follow a jointly multivariate Gaussian distribution, defined by a mean $\mu(x)$ and a covariance matrix $\Sigma(x)$. Each entry at location (i, j) in the covariance matrix $\Sigma(x)$ is defined by a kernel $\Sigma_{i,j} = K(x^{(i)}, x^{(j)})$, also known as a *covariance function*. The core idea is – if two data points are deemed similar by the kernel, the function outputs should be close, too. Making predictions with GP for unknown data points is equivalent to drawing samples from this distribution, via a conditional distribution of unknown data points given observed ones.

Check this post for a high-quality and highly visualization tutorial on what Gaussian Processes are.

Notation

P = Total no. of params

Let us consider a fully-connected neural networks with parameter θ, $f(.;\theta) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$. Layers are indexed from 0 (input) to L (output), each containing $n_0, \dots, n_L$ neurons, including the input of size n_0 and the output of size n_L. There are $P = \sum_{l=0}^{L-1} (n_l + 1) n_{l+1}$ parameters in total and thus we have $\theta \in \mathbb{R}^P$.

$(n_l \times n_{l+1}) + \underbrace{n_{l+1}}_{\text{↳ Bias}} = (n_l + 1) n_{l+1}$

The training dataset contains N data points, $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N$. All the inputs are denoted as $\mathcal{X} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and all the labels are denoted as $\mathcal{Y} = \{y^{(i)}\}_{i=1}^N$.

Now let's look into the forward pass computation in every layer in detail. For $l = 0, \dots, L-1$, each layer l defines an affine transformation $A^{(l)}$ with a weight matrix $\mathbf{w}^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and a bias term $\mathbf{b}^{(l)} \in \mathbb{R}^{n_{l+1}}$, as well as a pointwise nonlinearity function $\sigma(.)$ which is Lipschitz continuous.

$$\begin{aligned} A^{(0)} &= \mathbf{x} \\ \tilde{A}^{(l+1)}(\mathbf{x}) &= \frac{1}{\sqrt{n_l}} {\mathbf{w}^{(l)}}^\top A^{(l)} + \beta \mathbf{b}^{(l)} \quad \in \mathbb{R}^{n_{l+1}} & \text{; pre-activations} \\ A^{(l+1)}(\mathbf{x}) &= \sigma(\tilde{A}^{(l+1)}(\mathbf{x})) \quad \in \mathbb{R}^{n_{l+1}} & \text{; post-activations} \end{aligned}$$

Note that the *NTK parameterization* applies a rescale weight $1/\sqrt{n_l}$ on the transformation to avoid divergence with infinite-width networks. The constant scalar $\beta \geq 0$ controls how much effort the bias terms have.

All the network parameters are initialized as an i.i.d Gaussian $\mathcal{N}(0, 1)$ in the following analysis.

Neural Tangent Kernel

Neural tangent kernel (NTK) (Jacot et al. 2018) is an important concept for understanding neural network training via gradient descent. At its core, it explains how updating the model parameters on one data sample affects the predictions for other samples.

Let's start with the intuition behind NTK, step by step.

The empirical loss function $\mathcal{L} : \mathbb{R}^P \to \mathbb{R}_+$ to minimize during training is defined as follows, using a per-sample cost function $\ell : \mathbb{R}^{n_0} \times \mathbb{R}^{n_L} \to \mathbb{R}_+$:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$$

and according to the chain rule. the gradient of the loss is:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \underbrace{\nabla_\theta \overbrace{f(\mathbf{x}^{(i)}; \theta)}^{R^{n_L}}}_{\text{size } P \times n_L} \underbrace{\nabla_f \overbrace{\ell(f, y^{(i)})}^{\to R^+}}_{\text{size } n_L \times 1}$$

→ Derivative with $f \to R^{n_L}$ (output)

↳ Since derivative with $\theta \to R^P$

When tracking how the network parameter θ evolves in time, each gradient descent update introduces a small incremental change of an infinitesimal step size. Because of the update step is small enough, it can be approximately viewed as a derivative on the time dimension:

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^N \nabla_\theta f(\mathbf{x}^{(i)}; \theta) \nabla_f \ell(f, y^{(i)})$$

Again, by the chain rule, the network output evolves according to the derivative:

$$\boxed{\frac{df(\mathbf{x}; \theta)}{dt} = \frac{df(\mathbf{x}; \theta)}{d\theta} \frac{d\theta}{dt} = -\frac{1}{N} \sum_{i=1}^N \underbrace{\nabla_\theta f(\mathbf{x}; \theta)^\top \nabla_\theta f(\mathbf{x}^{(i)}; \theta)}_{\text{Neural tangent kernel}} \nabla_f \ell(f, y^{(i)})}$$

Transpose!. (To enable multiplication) ↳ NTK

Here we find the **Neural Tangent Kernel (NTK)**, as defined in the blue part in the above formula, $K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$: $\rightarrow f(x,\theta)$

x

$$\boxed{K(\mathbf{x}, \mathbf{x}'; \theta) = \nabla_\theta f(\mathbf{x}; \theta)^\top \nabla_\theta f(\mathbf{x}'; \theta)}$$

$$(P \times n_L)^T \cdot (P \times n_L)$$
$$= (n_L \times P) \cdot (P \cdot n_L)$$
$$= n_L \times n_L$$

where each entry in the output matrix at location $(m, n), 1 \leq m, n \leq n_L$ is:

$$\boxed{K_{m,n}(\mathbf{x}, \mathbf{x}'; \theta) = \sum_{p=1}^{P} \frac{\partial f_m(\mathbf{x}; \theta)}{\partial \theta_p} \frac{\partial f_n(\mathbf{x}'; \theta)}{\partial \theta_p}}$$

The "feature map" form of one input $\mathbf{x}$ is $\varphi(\mathbf{x}) = \nabla_\theta f(\mathbf{x}; \theta)$.

Infinite Width Networks

To understand why the effect of one gradient descent is so similar for different initializations of network parameters, several pioneering theoretical work starts with infinite width networks. We will look into detailed proof using NTK of how it guarantees that infinite width networks can converge to a global minimum when trained to minimize an empirical loss.

Connection with Gaussian Processes

Deep neural networks have deep connection with gaussian processes (Neal 1994). The output functions of a L-layer network, $f_i(\mathbf{x}; \theta)$ for $i = 1, \dots, n_L$, are i.i.d. centered Gaussian process of covariance $\Sigma^{(L)}$, defined recursively as:

$$\Sigma^{(1)}(\mathbf{x}, \mathbf{x}') = \frac{1}{n_0} \mathbf{x}^\top \mathbf{x}' + \beta^2$$

$$\lambda^{l+1}(\mathbf{x}, \mathbf{x}') = \begin{bmatrix} \Sigma^{(l)})(\mathbf{x}, \mathbf{x}) & \Sigma^{(l)})(\mathbf{x}, \mathbf{x}') \\ \Sigma^{(l)})(\mathbf{x}', \mathbf{x}) & \Sigma^{(l)})(\mathbf{x}', \mathbf{x}') \end{bmatrix}$$

$$\Sigma^{(l+1)}(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{f \sim \mathcal{N}(0, \lambda^{(l)})}[\sigma(f(\mathbf{x}))\sigma(f(\mathbf{x}'))] + \beta^2$$

Non Linearity

$$\rightarrow E(x,y) = E[(x - E(x))(y - E(y))]$$

Lee & Bahri et al. (2018) showed a proof by mathematical induction:

(1) Let's start with $L = 1$, when there is no nonlinearity function and the input is only processed by a simple affine transformation:

$$f(\mathbf{x}; \theta) = \tilde{A}^{(1)}(\mathbf{x}) = \frac{1}{\sqrt{n_0}} \mathbf{w}^{(0)\top} \mathbf{x} + \beta \mathbf{b}^{(0)}$$

$$\text{where } \tilde{A}_m^{(1)}(\mathbf{x}) = \frac{1}{\sqrt{n_0}} \sum_{i=1}^{n_0} w_{im}^{(0)} x_i + \beta b_m^{(0)} \quad \text{for } 1 \leq m \leq n_1$$

Since the weights and biases are initialized i.i.d., all the output dimensions of this network $\tilde{A}_1^{(1)}(\mathbf{x}), \dots, \tilde{A}_{n_1}^{(1)}(\mathbf{x})$ are also i.i.d. Given different inputs, the m-th network outputs $\tilde{A}_m^{(1)}(.)$

have a joint multivariate Gaussian distribution, equivalent to a Gaussian process with covariance function (We know that mean $\mu_w = \mu_b = 0$ and variance $\sigma_w^2 = \sigma_b^2 = 1$)

$$\begin{aligned}
\Sigma^{(1)}(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[\tilde{A}_m^{(1)}(\mathbf{x})\tilde{A}_m^{(1)}(\mathbf{x}')] \\
&= \mathbb{E}\Big[\Big(\frac{1}{\sqrt{n_0}}\sum_{i=1}^{n_0} w_{i,m}^{(0)} x_i + \beta b_m^{(0)}\Big)\Big(\frac{1}{\sqrt{n_0}}\sum_{i=1}^{n_0} w_{i,m}^{(0)} x'_i + \beta b_m^{(0)}\Big)\Big] \\
&= \frac{1}{n_0}\sigma_w^2 \sum_{i=1}^{n_0}\sum_{j=1}^{n_0} x_i {x'}_j + \frac{\beta\mu_b}{\sqrt{n_0}}\sum_{i=1}^{n_0} w_{im}(x_i + x'_i) + \sigma_b^2\beta^2 \\
&= \frac{1}{n_0}\mathbf{x}^\top\mathbf{x}' + \beta^2
\end{aligned}$$

(2) Using induction, we first assume the proposition is true for $L = l$, a l-layer network, and thus $\tilde{A}_m^{(l)}(.)$ is a Gaussian process with covariance $\Sigma^{(l)}$ and $\{\tilde{A}_i^{(l)}\}_{i=1}^{n_l}$ are i.i.d.

Then we need to prove the proposition is also true for $L = l + 1$. We compute the outputs by:

$$\begin{aligned}
f(\mathbf{x};\theta) &= \tilde{A}^{(l+1)}(\mathbf{x}) = \frac{1}{\sqrt{n_l}}\mathbf{w}^{(l)\top}\sigma(\tilde{A}^{(l)}(\mathbf{x})) + \beta\mathbf{b}^{(l)} \\
\text{where } \tilde{A}_m^{(l+1)}(\mathbf{x}) &= \frac{1}{\sqrt{n_l}}\sum_{i=1}^{n_l} w_{im}^{(l)}\sigma(\tilde{A}_i^{(l)}(\mathbf{x})) + \beta b_m^{(l)} \quad \text{for } 1 \le m \le n_{l+1}
\end{aligned}$$

We can infer that the expectation of the sum of contributions of the previous hidden layers is zero:

$\to \mu_w = 0$

$$\begin{aligned}
\mathbb{E}[w_{im}^{(l)}\sigma(\tilde{A}_i^{(l)}(\mathbf{x}))] &= \mathbb{E}[w_{im}^{(l)}]\mathbb{E}[\sigma(\tilde{A}_i^{(l)}(\mathbf{x}))] = \mu_w\mathbb{E}[\sigma(\tilde{A}_i^{(l)}(\mathbf{x}))] = 0 \\
\mathbb{E}[\big(w_{im}^{(l)}\sigma(\tilde{A}_i^{(l)}(\mathbf{x}))\big)^2] &= \mathbb{E}[{w_{im}^{(l)}}^2]\mathbb{E}[\sigma(\tilde{A}_i^{(l)}(\mathbf{x}))^2] = \sigma_w^2\Sigma^{(l)}(\mathbf{x}, \mathbf{x}) = \Sigma^{(l)}(\mathbf{x}, \mathbf{x})
\end{aligned}$$

Since $\{\tilde{A}_i^{(l)}(\mathbf{x})\}_{i=1}^{n_l}$ are i.i.d., according to central limit theorem, when the hidden layer gets infinitely wide $n_l \to \infty$, $\tilde{A}_m^{(l+1)}(\mathbf{x})$ is Gaussian distributed with variance $\beta^2 + \text{Var}(\tilde{A}_i^{(l)}(\mathbf{x}))$. Note that $\tilde{A}_1^{(l+1)}(\mathbf{x}), \dots, \tilde{A}_{n_{l+1}}^{(l+1)}(\mathbf{x})$ are still i.i.d.

$\tilde{A}_m^{(l+1)}(.)$ is equivalent to a Gaussian process with covariance function:

$$\begin{aligned}
\Sigma^{(l+1)}(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[\tilde{A}_m^{(l+1)}(\mathbf{x})\tilde{A}_m^{(l+1)}(\mathbf{x}')] \\
&= \frac{1}{n_l}\sigma\big(\tilde{A}_i^{(l)}(\mathbf{x})\big)^\top\sigma\big(\tilde{A}_i^{(l)}(\mathbf{x}')\big) + \beta^2 \quad \text{;similar to how we get } \Sigma^{(1)}
\end{aligned}$$

When $n_l \to \infty$, according to central limit theorem,

$$\boxed{\Sigma^{(l+1)}(\mathbf{x}, \mathbf{x}') \to \mathbb{E}_{f\sim\mathcal{N}(0,\Lambda^{(l)})}[\sigma(f(\mathbf{x}))^\top\sigma(f(\mathbf{x}'))] + \beta^2}$$

The form of Gaussian processes in the above process is referred to as the *Neural Network*

Gaussian Process (NNGP) (Lee & Bahri et al. (2018)).

Deterministic Neural Tangent Kernel

Finally we are now prepared enough to look into the most critical proposition from the NTK paper:

When $n_1, \dots, n_L \to \infty$ (network with infinite width), the NTK converges to be:

- **(1) deterministic at initialization, meaning that the kernel is irrelevant to the initialization values and only determined by the model architecture; and**
- **(2) stays constant during training.**

The proof depends on mathematical induction as well:

(1) First of all, we always have $K^{(0)} = 0$. When $L = 1$, we can get the representation of NTK directly. It is deterministic and does not depend on the network initialization. There is no hidden layer, so there is nothing to take on infinite width.

$W \to n_{L-1} \times n_L$

$x \to n_{L-1}$

$$f(\mathbf{x};\theta) = \tilde{A}^{(1)}(\mathbf{x}) = \frac{1}{\sqrt{n_0}}\mathbf{w}^{(0)\top}\mathbf{x} + \beta\mathbf{b}^{(0)}$$

$n_L \times 1$

$$K^{(1)}(\mathbf{x},\mathbf{x}';\theta) = \Big(\frac{\partial f(\mathbf{x}';\theta)}{\partial \mathbf{w}^{(0)}}\Big)^\top \frac{\partial f(\mathbf{x};\theta)}{\partial \mathbf{w}^{(0)}} + \Big(\frac{\partial f(\mathbf{x}';\theta)}{\partial \mathbf{b}^{(0)}}\Big)^\top \frac{\partial f(\mathbf{x};\theta)}{\partial \mathbf{b}^{(0)}}$$
$$= \frac{1}{n_0}\mathbf{x}^\top\mathbf{x}' + \beta^2 = \Sigma^{(1)}(\mathbf{x},\mathbf{x}')$$

FYI $\frac{\partial x^T w}{\partial x} = W^T$

Co vectors $(n \times 1)$ but $\frac{\partial x^T W}{\partial x} = W$ if $(1 \times n)$ Our case w is matrix refere 3rd case of

(2) Now when $L = l$, we assume that a l-layer network with $\tilde{P}$ parameters in total, $\tilde{\theta} = (\mathbf{w}^{(0)}, \dots, \mathbf{w}^{(l-1)}, \mathbf{b}^{(0)}, \dots, \mathbf{b}^{(l-1)}) \in \mathbb{R}^{\tilde{P}}$, has a NTK converging to a deterministic limit when $n_1, \dots, n_{l-1} \to \infty$.

$$\underline{K^{(l)}(\mathbf{x},\mathbf{x}';\tilde{\theta}) = \nabla_{\tilde{\theta}}\tilde{A}^{(l)}(\mathbf{x})^\top\nabla_{\tilde{\theta}}\tilde{A}^{(l)}(\mathbf{x}') \to K^{(l)}_\infty(\mathbf{x},\mathbf{x}')}$$

Assumption

☆ Note that $K^{(l)}_\infty$ has no dependency on θ.

Next let's check the case $L = l + 1$. Compared to a l-layer network, a $(l+1)$-layer network has additional weight matrix $\mathbf{w}^{(l)}$ and bias $\mathbf{b}^{(l)}$ and thus the total parameters contain $\theta = (\tilde{\theta}, \mathbf{w}^{(l)}, \mathbf{b}^{(l)})$.

The output function of this $(l+1)$-layer network is:

$$f(\mathbf{x};\theta) = \tilde{A}^{(l+1)}(\mathbf{x};\theta) = \frac{1}{\sqrt{n_l}}\mathbf{w}^{(l)\top}\sigma\big(\tilde{A}^{(l)}(\mathbf{x})\big) + \beta\mathbf{b}^{(l)}$$

And we know its derivative with respect to different sets of parameters; let denote $\tilde{A}^{(l)} = \tilde{A}^{(l)}(\mathbf{x})$ for brevity in the following equation:

$$\begin{aligned}
\nabla_{\mathbf{w}^{(l)}} f(\mathbf{x};\theta) &= \frac{1}{\sqrt{n_l}}\sigma(\tilde{A}^{(l)})^\top \quad \in \mathbb{R}^{1\times n_l} \\
\nabla_{\mathbf{b}^{(l)}} f(\mathbf{x};\theta) &= \beta \\
\nabla_{\tilde{\theta}} f(\mathbf{x};\theta) &= \frac{1}{\sqrt{n_l}}\nabla_{\tilde{\theta}}\sigma(\tilde{A}^{(l)})\mathbf{w}^{(l)} \\
&= \frac{1}{\sqrt{n_l}}\begin{bmatrix} \dot{\sigma}(\tilde{A}_1^{(l)})\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_1} & \dots & \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_1} \\ \vdots & & \\ \dot{\sigma}(\tilde{A}_1^{(l)})\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_{\tilde{P}}} & \dots & \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_{\tilde{P}}} \end{bmatrix}\mathbf{w}^{(l)} \quad \in \mathbb{R}^{\tilde{P}\times n_{l+1}}
\end{aligned}$$

→ Don't think this should have transpose

where $\dot{\sigma}$ is the derivative of σ and each entry at location $(p, m), 1 \le p \le \tilde{P}, 1 \le m \le n_{l+1}$ in the matrix $\nabla_{\tilde{\theta}} f(\mathbf{x};\theta)$ can be written as

$$\underline{\frac{\partial f_m(\mathbf{x};\theta)}{\partial\tilde{\theta}_p} = \sum_{i=1}^{n_l} w_{im}^{(l)}\dot{\sigma}(\tilde{A}_i^{(l)})\nabla_{\tilde{\theta}_p}\tilde{A}_i^{(l)}}$$

The NTK for this $(l+1)$-layer network can be defined accordingly:

$$\begin{aligned}
& K^{(l+1)}(\mathbf{x},\mathbf{x}';\theta) \\
=& \nabla_\theta f(\mathbf{x};\theta)^\top \nabla_\theta f(\mathbf{x};\theta) \\
=& \nabla_{\mathbf{w}^{(l)}} f(\mathbf{x};\theta)^\top \nabla_{\mathbf{w}^{(l)}} f(\mathbf{x};\theta) + \nabla_{\mathbf{b}^{(l)}} f(\mathbf{x};\theta)^\top \nabla_{\mathbf{b}^{(l)}} f(\mathbf{x};\theta) + \nabla_{\tilde{\theta}} f(\mathbf{x};\theta)^\top \nabla_{\tilde{\theta}} f(\mathbf{x};\theta) \\
=& \frac{1}{n_l}\Big[\sigma(\tilde{A}^{(l)})\sigma(\tilde{A}^{(l)})^\top + \beta^2
\end{aligned}$$

Hence summation

Cross Product $\nabla_{\tilde{\theta}}\sigma(A^{(l)})^T \cdot \nabla_{\tilde{\theta}}\sigma(A^{(l)})$

$$\begin{aligned}
& + \mathbf{w}^{(l)\top}\begin{bmatrix} \dot{\sigma}(\tilde{A}_1^{(l)})\dot{\sigma}(\tilde{A}_1^{(l)})\sum_{p=1}^{\tilde{P}}\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_p}\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_p} & \dots & \dot{\sigma}(\tilde{A}_1^{(l)})\dot{\sigma}(\tilde{A}_{n_l}^{(l)})\sum_{p=1}^{\tilde{P}}\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_p}\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_p} \\ \vdots & & \\ \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\dot{\sigma}(\tilde{A}_1^{(l)})\sum_{p=1}^{\tilde{P}}\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_p}\frac{\partial \tilde{A}_1^{(l)}}{\partial\tilde{\theta}_p} & \dots & \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\dot{\sigma}(\tilde{A}_{n_l}^{(l)})\sum_{p=1}^{\tilde{P}}\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_p}\frac{\partial \tilde{A}_{n_l}^{(l)}}{\partial\tilde{\theta}_p} \end{bmatrix}\mathbf{w}^{(l)}\Big] \\
=& \frac{1}{n_l}\Big[\sigma(\tilde{A}^{(l)})\sigma(\tilde{A}^{(l)})^\top + \beta^2 \\
& + \mathbf{w}^{(l)\top}\begin{bmatrix} \dot{\sigma}(\tilde{A}_1^{(l)})\dot{\sigma}(\tilde{A}_1^{(l)})K_{11}^{(l)} & \dots & \dot{\sigma}(\tilde{A}_1^{(l)})\dot{\sigma}(\tilde{A}_{n_l}^{(l)})K_{1n_l}^{(l)} \\ \vdots & & \\ \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\dot{\sigma}(\tilde{A}_1^{(l)})K_{n_l1}^{(l)} & \dots & \dot{\sigma}(\tilde{A}_{n_l}^{(l)})\dot{\sigma}(\tilde{A}_{n_l}^{(l)})K_{n_ln_l}^{(l)} \end{bmatrix}\mathbf{w}^{(l)}\Big]
\end{aligned}$$

where each individual entry at location $(m, n), 1 \le m, n \le n_{l+1}$ of the matrix $K^{(l+1)}$ can be written as:

$$K_{mn}^{(l+1)} = \frac{1}{n_l}\Big[\sigma(\tilde{A}_m^{(l)})\sigma(\tilde{A}_n^{(l)}) + \beta^2 + \sum_{i=1}^{n_l}\sum_{j=1}^{n_l} w_{im}^{(l)} w_{in}^{(l)}\dot{\sigma}(\tilde{A}_i^{(l)})\dot{\sigma}(\tilde{A}_j^{(l)})K_{ij}^{(l)}\Big]$$

When $n_l \to \infty$, the section in blue and green has the limit (See the proof in the previous section):

$$\boxed{\frac{1}{n_l}\sigma(\tilde{A}^{(l)})\sigma(\tilde{A}^{(l)}) + \beta^2 \to \Sigma^{(l+1)}}$$

and the red section has the limit:

$$\boxed{\sum_{i=1}^{n_l}\sum_{j=1}^{n_l} w_{im}^{(l)} w_{in}^{(l)} \dot{\sigma}(\tilde{A}_i^{(l)})\dot{\sigma}(\tilde{A}_j^{(l)}) K_{ij}^{(l)} \to \sum_{i=1}^{n_l}\sum_{j=1}^{n_l} w_{im}^{(l)} w_{in}^{(l)} \dot{\sigma}(\tilde{A}_i^{(l)})\dot{\sigma}(\tilde{A}_j^{(l)}) K_{\infty,ij}^{(l)}}$$

Later, Arora et al. (2019) provided a proof with a weaker limit, that does not require all the hidden layers to be infinitely wide, but only requires the minimum width to be sufficiently large.

Linearized Models

From the previous section, according to the derivative chain rule, we have known that the gradient update on the output of an infinite width network is as follows; For brevity, we omit the inputs in the following analysis:

$$\begin{aligned}
\frac{df(\theta)}{dt} &= -\eta \nabla_\theta f(\theta)^\top \nabla_\theta f(\theta) \nabla_f \mathcal{L} \\
&= -\eta \nabla_\theta f(\theta)^\top \nabla_\theta f(\theta) \nabla_f \mathcal{L} \\
&= -\eta K(\theta) \nabla_f \mathcal{L} \\
&= \underline{-\eta K_\infty \nabla_f \mathcal{L} \qquad \text{; for infinite width network}} \quad ☆
\end{aligned}$$

To track the evolution of θ in time, let's consider it as a function of time step t. With Taylor expansion, the network learning dynamics can be simplified as:

$$\boxed{f(\theta(t)) \approx f^{\text{lin}}(\theta(t)) = f(\theta(0)) + \underbrace{\nabla_\theta f(\theta(0))}_{\text{formally } \nabla_\theta f(\mathbf{x};\theta)|_{\theta=\theta(0)}} (\theta(t) - \theta(0))}$$

Such formation is commonly referred to as the *linearized* model, given $\theta(0)$, $f(\theta(0))$, and $\nabla_\theta f(\theta(0))$ are all constants. Assuming that the incremental time step t is extremely small and the parameter is updated by gradient descent:

$$\begin{aligned}
\theta(t) - \theta(0) &= -\eta \nabla_\theta \mathcal{L}(\theta) = -\eta \nabla_\theta f(\theta)^\top \nabla_f \mathcal{L} \\
f^{\text{lin}}(\theta(t)) - f(\theta(0)) &= -\eta \underline{\nabla_\theta f(\theta(0))^\top \nabla_\theta f(\mathcal{X};\theta(0))} \nabla_f \mathcal{L} \\
\frac{df(\theta(t))}{dt} &= -\eta K(\theta(0)) \nabla_f \mathcal{L} \\
\boxed{\frac{df(\theta(t))}{dt} = -\eta K_\infty \nabla_f \mathcal{L}} & \qquad \text{; for infinite width network}
\end{aligned}$$

↳ From above taylor expansion

NTK for θ(0)

Eventually we get the same learning dynamics, which implies that a neural network with infinite width can be considerably simplified as governed by the above linearized model (Lee & Xiao, et al. 2019).

In a simple case when the empirical loss is an MSE loss, $\nabla_\theta \mathcal{L}(\theta) = f(\mathcal{X};\theta) - \mathcal{Y}$, the dynamics of the network becomes a simple linear ODE and it can be solved in a closed form:

$$
\begin{aligned}
\frac{df(\theta)}{dt} &= -\eta K_\infty (f(\theta) - \mathcal{Y}) \\
\frac{dg(\theta)}{dt} &= -\eta K_\infty g(\theta) & & \text{; let } \underline{g(\theta) = f(\theta) - \mathcal{Y}} \\
\int \frac{dg(\theta)}{g(\theta)} &= -\eta \int K_\infty dt \\
\boxed{g(\theta) = Ce^{-\eta K_\infty t}}
\end{aligned}
$$

When $t = 0$, we have $C = f(\theta(0)) - \mathcal{Y}$ and therefore,

$$
\boxed{f(\theta) = (f(\theta(0)) - \mathcal{Y})e^{-\eta K_\infty t} + \mathcal{Y} = f(\theta(0))e^{-K_\infty t} + (I - e^{-\eta K_\infty t})\mathcal{Y}}
$$

And K_∞ does not depend on θ.

Lazy Training

People observe that when a neural network is heavily over-parameterized, the model is able to learn with the training loss quickly converging to zero but the network parameters hardly change. *Lazy training* refers to the phenomenon. In other words, when the loss $\mathcal{L}$ has a decent amount of reduction, the change in the differential of the network f (aka the Jacobian matrix) is still very small.

Let $\theta(0)$ be the initial network parameters and $\theta(T)$ be the final network parameters when the loss has been minimized to zero. The delta change in parameter space can be approximated with first-order Taylor expansion:

$$
\begin{aligned}
\hat{y} = f(\theta(T)) &\approx f(\theta(0)) + \nabla_\theta f(\theta(0))(\theta(T) - \theta(0)) \\
\text{Thus } & \boxed{\Delta\theta = \theta(T) - \theta(0) \approx \frac{\|\hat{y} - f(\theta(0))\|}{\|\nabla_\theta f(\theta(0))\|}}
\end{aligned}
$$

Still following the first-order Taylor expansion, we can track the change in the differential of f:

$$
\begin{aligned}
\nabla_\theta f(\theta(T)) &\approx \nabla_\theta f(\theta(0)) + \nabla^2_\theta f(\theta(0))\Delta\theta \\
&= \nabla_\theta f(\theta(0)) + \nabla^2_\theta f(\theta(0)) \frac{\|\hat{y} - f(\mathbf{x};\theta(0))\|}{\|\nabla_\theta f(\theta(0))\|} \\
\text{Thus } & \boxed{\Delta(\nabla_\theta f) = \nabla_\theta f(\theta(T)) - \nabla_\theta f(\theta(0)) = \|\hat{y} - f(\mathbf{x};\theta(0))\| \frac{\nabla^2_\theta f(\theta(0))}{\|\nabla_\theta f(\theta(0))\|}}
\end{aligned}
$$

Let $\kappa(\theta)$ be the relative change of the differential of f to the change in the parameter space:

Relative change in differential.

$$\boxed{\kappa(\theta = \frac{\Delta(\nabla_\theta f)}{\|\nabla_\theta f(\theta(0))\|} = \|\hat{y} - f(\theta(0))\| \frac{\nabla^2_\theta f(\theta(0))}{\|\nabla_\theta f(\theta(0))\|^2}}$$

Chizat et al. (2019) showed the proof for a two-layer neural network that $\mathbb{E}[\kappa(\theta_0)] \to 0$ (getting into the lazy regime) when the number of hidden neurons $\to \infty$. Also, recommend this post for more discussion on linearized models and lazy training.

Citation

Cited as:

> Weng, Lilian. (Sep 2022). Some math behind neural tangent kernel. Lil'Log. https://lilianweng.github.io/posts/2022-09-08-ntk/.

Or

```
@article{weng2022ntk,
  title   = "Some Math behind Neural Tangent Kernel",
  author  = "Weng, Lilian",
  journal = "Lil'Log",
  year    = "2022",
  month   = "Sep",
  url     = "https://lilianweng.github.io/posts/2022-09-08-ntk/"
}
```

References

[1] Jacot et al. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks." NeuriPS 2018.

[2]Radford M. Neal. "Priors for Infinite Networks." Bayesian Learning for Neural Networks. Springer, New York, NY, 1996. 29-53.

[3] Lee & Bahri et al. "Deep Neural Networks as Gaussian Processes." ICLR 2018.

[4] Chizat et al. "On Lazy Training in Differentiable Programming" NeuriPS 2019.

[5] Lee & Xiao, et al. "Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent." NeuriPS 2019.

[6] Arora, et al. "On Exact Computation with an Infinitely Wide Neural Net." NeurIPS 2019.

[7] (YouTube video) "Neural Tangent Kernel: Convergence and Generalization in Neural Networks" by Arthur Jacot, Nov 2018.

[8] (YouTube video) “Lecture 7 - Deep Learning Foundations: Neural Tangent Kernels” by Soheil Feizi, Sep 2020.

[9] “Understanding the Neural Tangent Kernel." Rajat’s Blog.

[10] “Neural Tangent Kernel."Applied Probability Notes, Mar 2021.

[11] “Some Intuition on the Neural Tangent Kernel." inFERENCe, Nov 2020.

foundation neural-tangent-kernel learning-dynamics

»

Generalized Visual Language Models