# Convolutional Neural Tangent Kernel

—

William Cappelletti

**Abstract**

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

**Semester Project**
**under the supervision of**
**Professor C. Hongler and Dr. F. Gabriel**

SFL Chair

**Autumn 2019**

Mathematics Section
MSc in Applied Mathematics 2019–2020

# Contents

# 1 Introduction

# 2 Neural Networks as Gaussian Processes

## 2.1 Tensor Programs

Our goal is to study how Artificial Neural Networks (ANN) behaves when their sizes become arbitrarily big and their parameters are initialized at random, following Glorot initialization [2].

To study such functions, Yang introduces the concept of *Tensor Program* in [6] and [7]. This framework is used to bind the sizes of parameters across different parts of the network, by explicitly binding to a line of the program introduction and usage of each parameter and non-linearity. In particular, *Tensor Programs* allow the input of vectors (**VecIn**) and matrices (**MatIn**) and their usage in linear combination of vectors (**LinComb**), matrix multiplication (**MatMul**), matrix transpose (**T**) and application of general (possibly nonlinear) functions (**Nonlin**). A more precise definition can be found in [6, Section 3], along with examples.

We can immediately see that Tensor Programs cover almost every standard Deep Learning architecture, and, in particular, they describe MLPs and CNNs. In spite of being more cumbersome of the usual algebraic formulation, this notation makes it easier to study ANN as we let their increase arbitrarily.

Intuitively, each program line binds a variable assignment and a dimension annotation, such that one can use each line result in other operations and keep track of the relations between dimensions. It follows that lines of type T, MatMul, LinComb, and Nonlin induce equality constraints on the dimensions of each line. We call *G-vars* the outputs of VecIn, MatMul and LinComb, and, similarly, *A-vars* those produced by MatIn and T and *H-vars* the results of Nonlin. Given a program $\pi$ and a possible set of additional dimensional constraints $\Lambda$, we can consider the smallest equivalence relation $\sim$ on G-vars such that $g \sim g'$ if their dimensions are constrained to be equal by $\Lambda$ or by some line of the program. With this relation, we can split the G-vars into classes and we call each class a *common dimension class* (CDC); we write $\mathfrak{C}$ the collection of all CDCs for a program $\pi$ and additional constraints $\Lambda$. The CDCs are the main instrument to understand the ANN behaviour when we let the network's dimensions increase, as all its elements scale together.

## 2.2 The Infinite Width Limit

We study how ANNs behave when we let their dimensions go to infinity, in what is called the Infinite Width Limit. Before we go deeper into theoretical results, we want to clarify what it means, for different architectures, to become infinitely wide. The easiest case is the MLP, in which all layers are fully connected. In this case, without additional constraints, the only dimensions that are bound are those of the input of a layer and the output of the previous one. Therefore, each layer gives its own CDC and the dimensions that go to infinity are the number of nodes in each layer (i.e the columns in each weight matrix and bias vector). This justifies the approach used in [3], where the authors make the layers sizes increase to infinity sequentially.

A different intuition arises when considering CNNs. In this case, by the definition of convolution, one cannot have the size of the filter to grow to infinity, as that would impose all layers to grow to infinity together and it would require an infinite size input, which is absurd. Therefore, in such a situation, the filter size stays the same, and the dimension that grows to infinity is the number of channels. To visualize what that means in terms of tensor programs, we can consider an image

with different channels. We take a vector across channels for every single pixel and we can see each filter application as first multiplying each pixel-vector by a weight matrix, giving as result a pixel-vector with a different number of channels, and then obtaining the preactivations for each coordinate of the new tensor as linear combinations of the latter pixel-vectors. Finally, one can apply the nonlinearity pointwise. For a complete example, we refer the reader to [6, Appendix B.6].

In the same way, we can express through tensor programs more complex architectures, such as Residual Neural Networks and other kinds of transformer, as shown in [6, Appendix B]. After we understand what dimensions go to infinity, we see that some similar behaviour arises, in spite of the architectures being significantly different. The first consequence is that in the *Infinite Width Limit* (IWL) all ANNs behave like *Gaussian Processes*.

Neal first proved, in [4], that a single-layer neural network with random parameters can converge in distribution to a Gaussian process as its width goes to infinity. [6] extends this result to any network that can be described by a tensor program.

We consider a sequence (in $t \in \mathbb{N}$) of dimensions $\{n^{lt}\}_{g^l \text{or} h^l} \cup \{n_1^{lt}, n_2^{lt}\}_{A^l}$ respecting the equivalence relation $\sim$ (defined in Section 2.1) in the program $\pi$, where $g^l$, $h^l$ and $A^l$ are, respectively, G, H, and A-vars appearing at line $l$ in $\pi$. At time $t$, we sample independently $A_{ij}^{lt} \sim \mathcal{N}(0, (\sigma^{lt})^2/n_2^{lt})$, for each $i$, $j$, for a set $\{\sigma^{lt}\}_{A^l}$. For each common dimension class $\mathfrak{c}$, we also sample independenty $g^{\mathfrak{c}_{\text{in}}t} \sim \mathcal{N}(\mu^{\mathfrak{c}_{\text{in}}t}, K^{\mathfrak{c}_{\text{in}}t})$ for each $i$. Here $\mathfrak{c}_{\text{in}}$ is the set of input G-vars in $\mathfrak{c}$, $g^{\mathfrak{c}_{\text{in}}t} = (g_i^{lt})_{g^l \in \mathfrak{c}_{\text{in}}}$, and $\mu^{\mathfrak{c}_{\text{in}}t} : \mathfrak{c}_{\text{in}} \to \mathbb{R}, K^{\mathfrak{c}_{\text{in}}t} : \mathfrak{c}_{\text{in}} \times \mathfrak{c}_{\text{in}} \to \mathbb{R}$ are specified mean and covaiance at time $t$. Thus, given $(\pi, \Lambda)$, the data $\{n^{\mathfrak{c}t}\}_{\mathfrak{c} \in \mathfrak{C}}$, $\{\sigma^{lt}\}_{A^l}$, $\{\mu^{\mathfrak{c}_{\text{in}}t}\}_{\mathfrak{c} \in \mathfrak{C}}$ and $\{K^{\mathfrak{c}_{\text{in}}t}\}_{\mathfrak{c} \in \mathfrak{C}}$ realize a random program $\pi(\{n^{\mathfrak{c}t}\}_{\mathfrak{c} \in \mathfrak{C}}, \{\sigma^{lt}\}_{A^l}, \{\mu^{\mathfrak{c}_{\text{in}}t}\}_{\mathfrak{c} \in \mathfrak{C}}, \{K^{\mathfrak{c}_{\text{in}}t}\}_{\mathfrak{c} \in \mathfrak{C}})$.

Furthermore, we assume that as $t \to \infty$, for all $\mathfrak{c}$, $\mathfrak{c}' \in \mathfrak{C}$:

1. $n^{\mathfrak{c}t}$ is increasing with $t$ and $n^{\mathfrak{c}t} \to \infty$.
2. $\lim_{t\to\infty} n^{\mathfrak{c}t}/n^{\mathfrak{c}'t} = \alpha_{\mathfrak{c},\mathfrak{c}'} \in (0, \infty)$, for some constant $\alpha_{\mathfrak{c},\mathfrak{c}'}$ depending only on $\mathfrak{c}' \in \mathfrak{C}$.
3. $\sigma^{lt} \to \sigma^{l\infty}$ for some finite $\sigma^{l\infty} > 0$ for each input A-var $A^l$.
4. $\mu^{\mathfrak{c}_{\text{in}}t} \to \mu^{\mathfrak{c}_{\text{in}}\infty}$ and $K^{\mathfrak{c}_{\text{in}}t} \to K^{\mathfrak{c}_{\text{in}}\infty}$ for some finite $\mu^{\mathfrak{c}_{\text{in}}\infty}$ and $K^{\mathfrak{c}_{\text{in}}\infty}$, and $\operatorname{rank} K^{\mathfrak{c}_{\text{in}}t} = \operatorname{rank} K^{\mathfrak{c}_{\text{in}}\infty}$ for all large $t$.

For this section main theorem, we need to restrict to a general class of nonlinearities, which we can intuitively think as meaning that the function is at most exponential.

**Definition 2.1.** For $\alpha > 0$, a function $\phi : \mathbb{R}^k \to \mathbb{R}$ is said to be *alpha-controlled* if for some $C$, $c > 0$, we have

$$|\phi(x)| \le \exp\left(C \sum_{i=1}^{k} |x_i|^\alpha + c\right) \tag{2.1}$$

for all $x \in \mathbb{R}^k$.

**Theorem 2.2** ([7])**.** *Consider ro correlations, but otherwise are asympthotically independent, unless they appear togheter in a LinComb. dimension costraints $\Lambda$ and a program $\pi$ without $T$ lines, i.e. no transpose allowed. Suppose the nonlinearities are $\alpha$-controlled for some $\alpha < 2$. Sample all input vars as explained beforhand (Glorot initialization). Then, for any $\mathfrak{c} \in \mathfrak{C}$ and any $\alpha$-controlled function $\phi : \mathbb{R}^{\mathfrak{c}} \to \mathbb{R}$, $\alpha < 2$,*

$$\frac{1}{n^{\mathfrak{c}t}} \sum_{i=1}^{n^{\mathfrak{c}t}} \phi(g_i^{\mathfrak{c}t}) \xrightarrow{a.s.} \mathbb{E}\phi(Z), \tag{2.2}$$

*where* $g_i^{\mathfrak{c}t} = (g_i^{lt})_{g^l \in \mathfrak{c}}$ *and* $\mathbb{R}^{\mathfrak{c}} \ni Z = (Z^g)_{g \in \mathfrak{c}} \sim \mathcal{N}(\mu^{\mathfrak{c}}, K^{\mathfrak{c}})$*, with* $\mu^{\mathfrak{c}}$ *and* $K^{\mathfrak{c}}$ *defined respectively in* (2.3) *and* (2.4)*.*

For any $\mathfrak{c} \in \mathfrak{C}$, we recursively define

$$\text{(mean)} \quad \mu^{\mathfrak{c}}(g^l) = \begin{cases} \mu^{\mathfrak{c}_{\text{in}}\infty}(g^l) & \text{if } g^l \in \mathfrak{c}_{\text{in}} \\ \sum_i a_{ji}^l \mu^{\mathfrak{c}}(g_i^j) & \text{if } g^l := \sum_i a_{ji}^l g_i^j \\ 0 & \text{if } g^l := A^k g^j \text{ or } g^l := A^k h^j \end{cases} \tag{2.3}$$

and recursively define

$$\text{(covariance)} \quad K^{\mathfrak{c}}(g^l, g^m) = \begin{cases} K^{\mathfrak{c}_{\text{in}}\infty}(g^l, g^m) & \text{if } g^l, g^m \in \mathfrak{c}_{\text{in}} \\ \sum_i a_{ji}^m K^{\mathfrak{c}}(g^l, g_i^j) & \text{if } g^m := \sum_i a_{ji}^m g_i^j \\ \sum_i a_{ji}^l K^{\mathfrak{c}}(g_i^j, g^m) & \text{if } g^l := \sum_i a_{ji}^l g_i^j \\ (\sigma^{k\infty})^2 \mathbb{E}_z[f^a(z) f^b(z)] & \text{if } g^l := A^k h^a, g^m := A^k h^b \\ 0 & \text{else} \end{cases} \tag{2.4}$$

where $h^a := f^a(g_1^j, \ldots, g_k^j), h^b := f^b(g_1^{j'}, \ldots, g_k^{j'})$ and $z \sim \mathcal{N}(\mu^{\mathfrak{c}}, K^{\mathfrak{c}})$. We also make branch 4 cover the case when $g^l := A^k g^a$ or $g^m := A^k g^b$ by "typecasting" $g^a$ to an H-var and setting $f^a = \text{id}$ (similarly for $g^b$). Note that, $f^a$ will ignore irrelevant components of a, and the expectations only depend on the entries of $\mu^{\mathfrak{c}}$ and $K^{\mathfrak{c}}$ that correspond to already-defined values, so this describes a valid recursion.

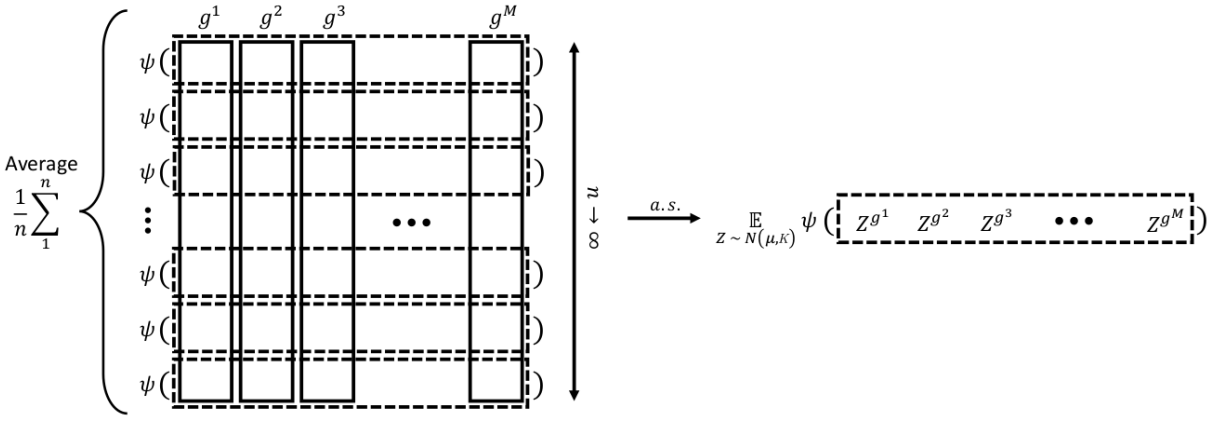


Figure 1: Illustration of Theorem 2.2, from [7]. If we suppose that $g^1, \ldots, g^M$ are all the G-vars in a CDC (i.e. $g^{\mathfrak{c}} = (g^1, \ldots, g^M)$), then, as the dimension $n$ increase, the empirical mean along the coordinates of the G-vars of an $\alpha$-controlled function $\psi$ converges to the expected value. Or, more intuitively, for each $i$, $(g_i^1, \ldots, g_i^M) \approx \mathcal{N}(\mu, K)$.

Intuitively, Theorem 2.2 states that $g_i^{\mathfrak{c}t} \overset{\text{d}}{\approx} \mathcal{N}(\mu^{\mathfrak{c}}, K^{\mathfrak{c}})$ for large $t$, iid for each $i$. This, jointly with the definitions of (2.3) and (2.4), means, roughly speaking, that the G-vars (preactivations, CFR Section 2.1) created from the same matrix $A^k$ have nonzero correlations, but otherwise are asymptotically independent unless they appear together in a LinComb.

## 2.3 Neural Tangent Kernel

Has we have seen in Section 2.2, each neaural network architecture is equivalent, in the IWL to a Gaussian Process. This implies that we can study the regression we perform with the ANN as a kernel method, once we understand which kernel corresponds to its training. In this section, $\{x^i\}_{i\in[N]} \subset \mathbb{R}^{\dim_{\text{in}}}$ denote the network inputs, $\theta \in \mathbb{R}^P$ contains all the parameters, distributed at initialization as in Section 2.2, and $f(\theta, x)$ is the output of the network, which, for notation simplicity, we suppose unidimensional.

As in [3] and [1], we consider a training dataset $\{(x^i, y^i)\}_{i=1}^N \subset \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}$ and we train the neural network by minimizing a convex loss function

$$l(\theta) = \sum_{i=1}^{N} \mathcal{L}(f(\theta, x^i), y^i).$$

We update the parameters at step $t$ using gradient descent with an infinitesimally small learning rate $\frac{d\theta(t)}{dt} = -\nabla l(\theta(t))$. With this setup, if we consider $l$ to be the mean squared error, by a simple differentiation [1, Lemma 3.1] we see that the outputs of the network $u(t) = (f(\theta(t), x^i))_{i\in[N]}$ evolve according to

$$\frac{du(t)}{dt} = -\operatorname{NTK}(f_t) \cdot (u(t) - y), \tag{2.5}$$

where $\operatorname{NTK}(f_t)$ is the $n \times n$ positive semidefinite matrix given by the *Neural Tangent Kernel*.

**Definition 2.3.** With the notation previously introduced, we define the *Neural Tangent Kernel* $\operatorname{NTK}(\cdot, \cdot)$ as the kernel whose values on points $x, x'$ are given by

$$\operatorname{NTK}(x, x') = \left\langle \frac{\partial f(\theta, x)}{\partial \theta}, \frac{\partial f(\theta, x')}{\partial \theta} \right\rangle. \tag{2.6}$$

As the values of $\theta$ are randomly initialized, we also define

$$\operatorname{NTK}_\infty(x, x') = \mathbb{E}_\theta[\operatorname{NTK}(x, x')]. \tag{2.7}$$

Finally, for any kernel $K$, we abuse the notation and, when the context make it clear, use $K$, or specifically $NTK$, to define the matrix whose entries $i, j$ are given by $K(x^i, x^j)$.

As we can see, the NTK depends on the parameters of the network and thus it is random at initialization and varies during training. Nonetheless, in [3], the authors prove that, if the ANN is an MLP, the NTK converges in the IWL to $\operatorname{NTK}_\infty$, which is ideed deterministic. Therefore, the dynamic expressed in (2.5) is identical to that of *kernel regression* under gradient flow.

It turns out that this idea is even more general. In fact, in [3] Jacot et al. prove that the network function evolves along the kernel gradient given by the Neural Tangent Kernel for any convex loss function. We conjecture that this result holds true for any deep learning architecture.

## 2.4 Kernel Regression

Before moving to the experimental part, we remind the rider some concepts about *kernel regression*. Suppose we have a kernel function $K(\cdot, \cdot)$ and a training dataset $\{(x^i, y^i)\}_{i=1}^N \subset \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}$, kernel regression aims to approximate the underlying function with the linear combination

$$f(\cdot) = \sum_{i=1}^{N} \alpha^i K(\cdot, x^i). \tag{2.8}$$

The coefficients are estimated to minimize the loss function over prediction points

$$l(\alpha) = \mathcal{L}(f(x_*), y_*),$$

and in practice this minimization is carried over the training dataset

$$\widehat{\alpha} = \arg\inf \left\{ \sum_{i=1}^{N} \mathcal{L} \left( \sum_{j=1}^{N} \alpha_j K(x^i, x^j), y^i \right) : \alpha_1, \ldots, \alpha_N \in \mathbb{R} \right\}. \tag{2.9}$$

### 2.5 Multidimensional output

All we stated before about neural networks and kernels generalize immediately to the multidimensional case upon noting that a function $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is equivalent to a function $\tilde{f} : \mathbb{R}^{d_{\text{in}}} \times [d_{\text{out}}] \to \mathbb{R}$ with $\tilde{f}(x, k) = f_k(x)$. This said, we can generalize kernel regression, with each entry of the kernel $K(x, x')$ now being a matrix instead of a scalar and $\alpha$ being a vector, such that the vector-matrix product in (2.8) produces indeed a vector output. In this case, the kernel matrix $K$ is a block matrix, where each block is given by the matrix $K(x^i, x^j)$.

For the Neural Tangent Kernel, this means that

$$[NTK(x^i, x^j)]_{k,k'} = \left\langle \frac{\partial f_k(\theta, x^i)}{\partial \theta}, \frac{\partial f_{k'}(\theta, x^j)}{\partial \theta} \right\rangle, \tag{2.10}$$

with $k, k' \in [d_{\text{out}}]$. An interesting point of the multidimensional approach is that, as we can see in (2.10), the kernel also encodes information between different output coordinates, i.e. $k \neq k'$.

## 3 NTK implementation

In order to practically test the theoretical results on different deep learning architectures, we choose to work using the efficient and popular library *Pytorch* [5], which is highly optimized for tensorial calculus.

## 4 Experimental results

## References


[1] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[3] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[4] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[6] G. Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

[7] G. Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *arXiv preprint arXiv:1910.12478*, 2019.