# THE ASYMPTOTIC STRUCTURE OF DEEP NEURAL NETWORKS

by **João L. Costa***

Abstract.—Deep Neural Networks (DNNs) are the main concept at the center of the artificial intelligence revolution we are experiencing. However, some of the reasons behind their effectiveness (for instance, why do they seem to provide "good" solutions, determined by simple optimization algorithms?), as well as the causes of their limitations (for instance, why are they so parameter and data expensive?), remain somewhat unclear. Therefore, a theoretical/mathematical clarification of these issues would be welcomed and, in principle, might help us in the construction of a new generation of interpretable, safer, sustainable and, consequently, more reliable AI models.

With that in mind, a mathematical approach that has provided some relevant insights is the study of the asymptotic structure of DNNs.

In this article, we will start by introducing the basics of DNNs, followed by a presentation of some results concerning the study of the large width limit of these models and a discussion of the implications that such results have in our understanding of supervised machine learning with DNNs.

## 1 Introduction

The artificial intelligence revolution we are experiencing, with remarkable breakthroughs in natural language processing, computer vision, natural sciences, among others, is being fueled by the use of unprecedented computational power and data consumption, but, at the core of such developments, is the concept of artificial Deep Neural Networks (DNNs). Mathematically, DNNs correspond to specific parameterized families of functions (explicit examples will be provided below) that were originally designed as (extremely simplified) models of biological brains.

DNNs are trained by tuning their parameters in order to minimize a chosen loss function that measures the ability of a network to reproduce a given relation between training inputs and labels/targets [1]; for instance, in the famous MNIST dataset, the inputs are $28 \times 28$ matrices of integers between 0 and 255, encoding a low resolution gray scale image of handwritten digits, and the labels are integers between 0 and 9, identifying the corresponding digit.

It should be stressed, from the beginning, that the final goal of the described machine learning procedure goes beyond the simple "memorization" of the training set, meaning attaining near zero loss in the training set; in fact, this could be achieved by a simple dictionary. The quality of the trained model should be evaluated in terms of its generalization capabilities, meaning its ability to perform well (i.e., have a small loss), in test sets, composed of data that hasn't been used in the training/optimization procedure; for instance, a model trained on MNIST generalizes well if it correctly classifies a high percentage of handwritten digits in images that it hasn't "seen before".

Recently, the increase in the size of deep neural networks, both in the number of trainable parameters and the amount of training data resources, has been proportional to the astonishing success of using DNNs in practical applications [2]. This motivates the study of the asymptotic structure of DNNs, when the number of parameters tends to infinity. As we

[1] Strictly speaking, what we are describing is a problem in the subfield of supervised learning, where a labeled data set, consisting of pairs of training inputs and labels/targets, known as training set, is provided a priori. DNNs are also relevant in self-supervised and reinforcement learning problems.

[2] One should however note, that this trend, plenty of times extrapolated into optimistic "scaling laws", as started to show signs of slowing down [20].

* Departamento de Matemática (ISCTE-IUL) and
Center for Mathematical Analysis, Geometry and Dynamical Syatems (IST-UL)

will see, there are two basic dimensions in a DNN: its width, meaning the number of neurons per layer, and its depth, encoding the number of layers. Here we will be interested in the large width limit of DNNs, which allows us, for instance, to shed some light into the following fundamental questions:

**Q1:** (*Memorization capacity*) why is it that the training of DNNs, via gradient descent (a standard optimization technique to be described below), is, in various applications, able to achieve (nearly) zero loss, even though the loss function is highly non-convex?

**Q2:** (*Generalization performance*) why is it that some trained networks demonstrate good generalization performance? Moreover, how does this happen even in the overparametrized regime, where the number of parameters is much larger than the amount of data, in blatant contrast with the expectations of classical statistical learning theory?

**Q3:** (*Scaling laws*): do we really expect the performance of DNNs to keep on improving with the increase in the number of parameters?

**Q4:** (*Hyper-parameters tuning*) contrary to some oversimplifying narratives that create unreasonable expectations, in general, it isn't easy to train a DNN. In fact, typically, a successful training process involves, beyond the data pre-processing, an overwhelming number of choices, that go from the choice of architecture, including the number of layers and the number of neurons per layer, to the choice of parameter initialization, among many others. Even though the process has become increasingly more robust, the referred choices often follow educated guesses, based on previous experience and heuristic theoretical arguments, and a strenuous and expensive process of trial and error. It would be relevant to be able to obtain some guidance for these procedures in the form of mathematical results.

Still regarding the challenges facing the training of DNNs we haven't yet mentioned, but are not forgetting, what is arguably the most important one: fighting the tendency of large models to overfit, specially in a context of data scarcity. Overfitting corresponds to the situation when training leads to a "small" loss in the training set, but a "large" loss in the test set, i.e., a situation where the model memorizes the training set, without generalizing well to the test set. Various techniques [9, Chapter 7] have been devised to tame the tendency for overfitting, which range from adapting classical ideas, such as the inclusion of a regularization term in the loss, to the somewhat crude and self-explanatory idea of early-stopping, and a surprisingly clever and efficient strategy known as *dropout*, whose trademark feature is the freezing of randomly chosen components of the model in each step of training. Unfortunately, do to spacetime constraints, we won't be discussing these important topics any further here.

## 2 Deep Learning: the theoretical minimum

In this section we will describe some examples of artificial Neural Network models and will take the opportunity to clarify what is meant by learning through gradient descent.

### 2.1 Multilayer Perceptrons

We will start with the first historical example of an artificial neuron [14], which was originally designed by McCulloch and Pitts with the grand goal of being the fundamental unit for neural computations and, later, named accordingly as the Perceptron, by Rosenblatt [16]. This corresponds simply to the parameterized family of functions $f_\theta : \mathbb{R}^n \to \mathbb{R}$ defined by

$$f_\theta(x) = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{1}$$

where the model parameters, known as weights $w_i$ and bias $b$, are collectively stored in $\theta = \{w_1, \cdots, w_n, b\}$. The function $\phi$ is known as the activation function or non-linearity; originally it was chosen to be the Heaviside function $\phi(z) = 1_{[0,+\infty[}(z)$, but since this function has an almost everywhere vanishing derivative it renders the model untrainable in a context of gradient based optimization, that we will discuss soon. Other examples of commonly used activations are the sigmoid function $\phi(z) = 1/(1 + e^{-z})$ and the ReLU (Rectified Linear Unit) $\phi(z) = \max\{0, z\}$.

Rosenblat also provided a clever and geometrically transparent learning algorithm that allows Perceptrons to classify linearly separable datasets. Unfortunately, this is all Perceptrons can do. Nonetheless, we can easily extend this class of functions into a far more expressive one: i) first we stack Perceptrons on top of each other creating a Dense Layer, which is a function $\Phi : \mathbb{R}^n \to \mathbb{R}^m$, defined by

$$\Phi(x) = \phi\left(\frac{1}{\sqrt{n}} W x + b\right),$$

with weight matrix $W \in \mathbb{R}^{m\times n}$, bias vector $b \in \mathbb{R}^m$, and where the activation function acts entrywise, i.e. $\phi([z_1 \cdots z_m]^t) := [\phi(z_1) \cdots \phi(z_m)]^t$; ii) then we can compose several of these layers to obtain a Multilayer

Perceptron (MLP)[3] which is the class of functions

$$f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_{\ell+1}}, \tag{2}$$

defined recursively by:

$$\begin{aligned} y^{(0)} &= x \\ y^{(1)} &= \phi\left(\frac{1}{\sqrt{n_0}} W^{(1)} y^{(0)} + b^{(1)}\right) \\ y^{(2)} &= \phi\left(\frac{1}{\sqrt{n_1}} W^{(2)} y^{(1)} + b^{(2)}\right) \\ &\vdots \\ y^{(\ell+1)} &= \frac{1}{\sqrt{n_\ell}} W^{(\ell+1)} y^{(\ell)} + b^{(\ell+1)} . \end{aligned}$$

The final layer, layer $\ell + 1$, is the output layer, the previous layers, if they exist, are called hidden layers. For each layer we have a matrix of weights $W^{(i)} \in \mathbb{R}^{n_{i+1} \times n_i}$ and a vector of biases $b^{(i)} \in \mathbb{R}^{n_{i+1}}$. As before, we collect all learnable parameters in $\theta = \{W^{(i)}, b^{(i)}\}_{i=1,\cdots,\ell+1}$ . In some applications it is also important to apply a non-linearity $\phi_{out}$ to the output layer, which is typically different from the inner activations $\phi$; here we'll be mostly focused on the case $\phi_{out}(z) = z$, as described. We also say that each layer has width $n_i$, which is sometimes referred as the number of neurons/units in the layer, and say that the network has depth $\ell + 1$; so, the "deep" in "deep learning" is simply a reference to the number of consecutive layers that are composed in a given DNN.

The normalization factors $n_\kappa^{-1/2}$ are used, in particular, to tame overflow issues when $\sum_\kappa n_\kappa \gg 1$. The way they were introduced here is not standard in the machine learning literature, where normalization is carried out in the initialization of the parameters, but has the advantage of making the infinite width limits, $n_\kappa \to \infty$, more transparent.

This construction immediately pays off. While Perceptrons (or a single Dense Layer) were highly restricted in their expressiveness, sufficiently wide MLPs, with at least one hidden layer ($\ell \geq 1$), can approximate any function, in relevant classes. Results formalizing this kind of statement are known as Universal Approximation Theorems. An example of such a statement is the following

Theorem 1.— Let $\phi \in C(\mathbb{R})$. The set

$$\text{span}\{\phi(w \cdot x + b) \mid w \in \mathbb{R}^n, b \in \mathbb{R}\}$$

is dense in $C(\mathbb{R}^n)$ if and only if $\phi$ is not a polynomial.

If a single hidden layer is enough to render DNNs universal approximators why go any deeper? This is a deep question (pun not intended). From a practical standpoint, it is clear that deepness is indispensable to solve complex problems [4], and the typical heuristic explanation for this goes something like this: deeper layers ($\ell \gg 1$) learn more abstract features [5], which are built on top of earlier, more tangible features, and this level of abstraction is required to handle sophisticated problems such as natural language processing and computer vision [6].

At a mathematical level the importance of deepness can be connected to notions of efficiency and expressivity. A simple observation along this lines is the following: consider the function $f(x) = 2\max\{0, x\} - 4\max\{0, x - 1/2\}$; in the language of DNNs, this model has a single hidden layer, with 2 neurons and ReLU activation. In a different dialect, it defines a piecewise linear function with 2 affine linear pieces. The $\ell$-fold composition of $f$ with itself, $f_\ell = f \circ \cdots \circ f$, is a depth $\ell + 1$ MLP with the capacity to produce $2^\ell$ affine linear pieces, using only $2\ell$ (hidden) neurons; achieving the same with a single hidden layer ReLU network would require $2^{\ell-1}$ neurons [17].

## 2.2 Training

In supervised machine learning there are three main ingredients: a training set $\{(X_i, Y_i)\}_{i=1}^N$, consisting of inputs $X_i$ and targets/labels $Y_i$; a model class $M$, here we are interested in DNNs so we consider $M = \{f_\theta \mid \theta \in \mathbb{R}^P\}$, with, for instance, the $f_\theta$ as defined in (2); and a loss function $\mathscr{L} : \mathbb{R}^P \to \mathbb{R}_0^+$, where $P$ is the number of parameters of the chosen class of DNNs; for simplicity let us assume that the loss is the squared error loss, given by

$$\mathscr{L}(\theta) := \frac{1}{2} \sum_{i=1}^{N} \left(Y_i - f_\theta(X_i)\right)^2 . \tag{3}$$

In this context, training corresponds to minimizing the loss, i.e. solving the problem $\theta^* = \text{argmin}\, \mathscr{L}(\theta)$.

[3] This architecture goes by various other names, such as *dense feedforward* or *fully conencted* neural networks.
[4] For instance, according to itself, the GPT-3 architecture has 96 transformer layers.
[5] This is a somewhat vague notion, that, in some contexts, can be defined as the component functions of each hidden layer. Also, it is the data that has "features" and, in that regard, we should talk about "feature-detectors", or something along those lines, when talking about models.
[6] Interesting empirical evidences in favor of this argument can be obtained by probing the responses of each layer in trained DNNs [7, Section 9.4.2].

When training DNNs, the typical minimization procedures used are based on gradient descent, meaning that the training trajectory, in parameter space, is determined by

$$\dot{\theta}(t) = -\nabla_\theta \mathscr{L}(\theta(t)), \tag{4}$$

with parameter initialization $\theta_0 = \theta(0)$ chosen according to some fixed probability distribution. One should note that, in practice, one uses discretized versions of gradient descent, with distinct degrees of sophistication: these can include batch learning, second order information and dynamic normalization factors, designed to handle memory issues, to allow convergence speedup and promote training stability.

It is well known that under quite general assumptions, gradient descent converges to a local minimum. Nonetheless, since the loss landscape is highly non-convex and can often include a large number of local minima, it is not at all clear, a priori, why the described method should lead to "good" solutions as measure by the smallness of the final loss.

Gradient descent also requires the computation of the derivatives of the loss with respect to the model parameters. This isn't done using a numerical computational scheme, such as finite differences. In fact, this is achieved through the explicit computation of recursive formulas, in terms of elementary functions, obtained by using the chain rule; in retrospect, the most natural of tools in view of the compositional nature of DNNs. This procedure is known as back-propagation, since the referred recursion starts from the output layer, layer $\ell + 1$, and then propagates the information to the earlier layers, in reversed order.

### 2.3 A brief overview of other architectures.

MLPs still play a key role in deep learning, but they are not the only available choice of neural network architecture. In practice, we should think of them as a useful and widely used component in a plentiful menu of architectures. More generally, we can define DNNs as functions $\Psi : \mathbb{D}_1 \to \mathbb{D}_2$ which are the composition of several *layers* $\Phi^{(\kappa)} : \mathbb{D}_{\kappa_1} \to \mathbb{D}_{\kappa_2}$, resulting in

$$\Psi = \Phi^{(L)} \circ \cdots \circ \Phi^{(1)} .$$

The domains $\mathbb{D}$ are typically real vector spaces, including $\mathbb{R}^n$, matrix spaces or higher order tensor spaces, but can also be integer based spaces such as $\mathbb{Z}_k^n$, among others.

In this setting, we can construct new DNN architectures, by defining new layers and choosing appropriate ways to combine them. Here are a couple of famous examples:

A convolutional layer is, as the name suggests, obtained by applying convolution operations; in a somewhat schematical way $\Phi(x) = W * x$, where $W$ is a learnable convolution kernel, that can, for instance, be used to learn how to detect translational invariant features of the input $x$. A convolutional neural network (CNN), is typically the result of composing several convolutional layers, intertwined with another type of layers known collectively as *pooling layers*, designed to downsample the size of intermediate outputs, and, to finish, an MLP. CNNs have had a tremendous impact in computer vision (see [9, Chapter 9] and references therein, for more information).

An embedding layer is, in some contexts, a map from the integers, encoding, for instance, the position of words (or, more generally, tokens) in a dictionary, to a real vector space $\mathbb{R}^{d_e}$. A self-attention layer, picks up a collection of these embeddings, compiled as columns of a matrix $E$, and computes an attention score through the now famous formula

$$\text{Attention}(E; W_Q, W_K, W_V) = \text{softmax}\left(\frac{(W_Q E)^t W_K E}{\sqrt{d_e}}\right) W_V E \,, \tag{5}$$

where the $W$'s are matrices of learnable parameters and $\text{softmax}_j(z_1, \dots, z_m) := e^{z_j} / \sum_i e^{z_i}$ is used column-wise. The famous *transformer* architecture [18], the "T" in GPT, is an intricate combination of embedding layers, attention layers, normalization layers and, once again, MLPs.

## 3 The large width limit

### 3.1 Shallow networks

It is instructive to start by analyzing the simplest model class, within MLPs (2): shallow networks ($\ell = 1$), with a single input ($n_0 = 1$) and a single output ($n_2 = 1$). In such case, MLPs reduce to the class of functions $f_\theta : \mathbb{R} \to \mathbb{R}$ of the form

$$f_\theta(x) = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} \left( W_j^{(2)} \, \phi\left( W_j^{(1)} x + b_j^{(1)} \right) + b_j^{(2)} \right) . \tag{6}$$

Notice that here we didn't follow the definition 2 to the letter. Instead we've added extra output bias terms, the $b_j^{(2)}$, one for each hidden neuron. This is sometimes convenient since it allows us to write our

network as a sum of perceptrons

$$f_\theta(x) = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} p_j(x) \, . \tag{7}$$

An immediate consequence of this form is the following: if we initialize each set of parameters in a independent and identically distributed (iid) fashion, with finite variance and zero mean [7], then, for each $x \in \mathbb{R}$, the $p_j(x)$ are also iid random variables with zero mean and finite variance, and, therefore, the Central Limit Theorem guarantees that, in the infinite width limit, $n \to \infty$, $f_\theta(x)$ converges, in distribution, to a random variable with Gaussian distribution $N(0, \Sigma_{xx})$, where $\Sigma_{xx} := \mathbb{E}[p^2(x)]$. This provides a clear description of the measure, in the function space $\mathrm{Map}(\mathbb{R}, \mathbb{R})$, induced by a standard parameter initialization via the map $\mathbb{R}^P \ni \theta \mapsto f_\theta \in \mathrm{Map}(\mathbb{R}, \mathbb{R})$; nonetheless, this was considered to be "disappointing", from the time of its discovery [15], since wide neural networks were being "forced" into a well known Gaussian regime, instead of fulfilling the expectation of being flexible enough to develop a wider range of data driven features.

The previous result concerns the expressivity of wide (but shallow) neural networks at initialization. Let's now see what happens during training, via gradient descent (4). Instead of studying the training dynamics in parameter space directly, a clearer path turns out to be traced by the evolution of outputs $f_{\theta(t)}(x)$, which is determined by (4) together with

$$\partial_t f_{\theta(t)}(x) = \sum_{l=1}^{N} \Theta_n(x, X_l; \theta(t)) \left( Y_l - f_{\theta(t)}(X_l) \right) \, , \tag{8}$$

where, after ordering the parameters $\theta = \{\theta_1, \dots, \theta_{4n}\}$,

$$\Theta_n(z_1, z_2; \theta) := \sum_{i=1}^{4n} \frac{\partial f_\theta}{\partial \theta_i}(z_1) \frac{\partial f_\theta}{\partial \theta_i}(z_2) \, , = \frac{1}{n} \left( \sum_{j=1}^{n} \sum_{i=1}^{4n} \frac{\partial p_j}{\partial \theta_i}(z_1) \frac{\partial p_j}{\partial \theta_i}(z_2) \right) ,$$

is known as the (empirical) Neural Tangent Kernel (NTK) [11]. At this point, it is the Law of Large Numbers that controls the asymptotics by ensuring that, as $n \to +\infty$,

$$\Theta_n(z_1, z_2; \theta) \to \Theta_\infty(z_1, z_2) := \mathbb{E}\left[ K(z_1, z_2) \right] ,$$

almost surely (a.s.), where $K(z_1, z_2)$ is the random variable

$$K(z_1, z_2) := \sum_j \frac{\partial p_1}{\partial \theta_j}(z_1) \frac{\partial p_1}{\partial \theta_j}(z_2) \, .$$

So we see that $\Theta_\infty$, the infinite width NTK, is a.s. constant in parameter space. This is a fundamental observation, first made explicitly in [11], with remarkable consequences. Loosely speaking, it implies that the "infinite width network" follows a linear learning evolution, obtained by formally taking the $n \to \infty$ limit in (8). In rigor, we can't define the infinite width network as a function, by taking the limit, $n \to \infty$, in (6), since this limit diverges [8] with probability one. To circumvent this difficulty, we take inspiration from the referred formal limit in (8), and define the *infinite width network*, at training time $t \geq 0$, as the solution to the initial value problem

$$\begin{cases} \partial_t f_\infty(x, t) = \\ \qquad - \sum_{j=1}^{N} \Theta_\infty(x, X_j)(f_\infty(X_j, t) - Y_j) \\ f_\infty(x, 0) = f_{\theta(0)}(x) \, . \end{cases} \tag{9}$$

Is is important to stress that, strictly speaking, the infinite width network is not a neural network and it isn't even a parametric model — "parameters are washed/integrated away by the limit theorems". In view of the constancy of the NTK in parameter space, (9) is a closed linear system of Ordinary Differential Equations (ODEs), that can be integrated to show that training, in the infinite width regime, converges to

$$\lim_{t \to +\infty} f_\infty(x, t) = f_\infty(x, 0) - \sum_{l,m=1}^{N} \Theta_\infty(x, X_l)(\Theta_\infty)^{-1}_{lm}(f_\infty(X_m, 0) - Y_m), \tag{10}$$

where $(\Theta_\infty)^{-1}_{lm}$ is the $(l, m)$ component of the inverse of the matrix $\Theta_\infty(X_i, X_j)$. Formula (10) is an instance of Kernel Regression, another well known technique in machine learning; consequently, the previous result is sometimes summarized by saying that, in the infinite width limit, neural networks enter a kernel regime.

We can now return to finite width neural networks. By using a bootstrap/continuity argument it can be shown that, for sufficiently wide networks, $n \gg 1$, the non-linear learning dynamics defined by both (4) and (8), can be approximated by the infinite width linear evolution. More precisely we can show

[7] This conditions can be easily relaxed.
[8] Has divergent superior and inferior pointwise limits.

that

$$\left|\Theta_n(z_1, z_2; \theta(t)) - \Theta_\infty(z_1, z_2)\right| \leq C \frac{(\log n)^{1/2+\epsilon}}{\sqrt{n}} , \tag{11}$$

and that

$$\left|f_{\theta(t)}(x) - f_\infty(x, t)\right| \leq C \frac{(\log n)^{1/2+\epsilon}}{\sqrt{n}} , \tag{12}$$

for all $t \geq 0$ and all $|x|, |z_1|, |z_2| \leq 1$. The previous, in particular, imply that, for sufficiently large $n$, the training outputs converge exponentially to their labels:

$$|f_{\theta(t)}(X_l) - Y_l| \leq C e^{-(\lambda_{\min} - \epsilon)t} , \tag{13}$$

where $\lambda_{\min}$ is the minimum eigenvalue of the matrix $\Theta_\infty(X_i, X_j)$, which, for now, we are assuming to be positive.

We should stop for a minute to contemplate what was achieved, even if restricted to the shallow network case: First, it should be clear that (13) implies that, for sufficiently wide networks, the loss (3) converges to zero, a global minimum; this corresponds to the memorization problem discussed in the introduction (see **Q1**). Secondly, Equations (12) and (13) provide control over the trained model in terms of an explicit Kernel Regression, that in turn depends only on the data and initialization; in particular, the generalization performance (see **Q2**) of the learned model is also codified in these and can be further understood by relying on the learning theory for Kernel Regression (see for instance [2, 13]).

As the more cautious reader surely has noticed, some of the presented results are conditional on the assumption that the matrix $\Theta_\infty(X_i, X_j)$ is (strictly) positive definite. This turn out to be the case under very general assumptions: basically one just needs to assume that all training inputs $X_i$ are distinct and that the activation function is continuous, almost everywhere differentiable and non-polynomial [4].

### 3.2 Deep networks

It turns out that most of the results discussed in the previous section generalize to the context of DNNs, so, hopefully, the pedagogical incursion through the framework of shallow networks was fruitful. In this section we will briefly sketch how this generalization proceeds by focusing of the construction of the NKT in the context of DNNs.

With that in mind, let us drop the restrictions made in the previous section and consider neural networks, more precisely MLPs, of arbitrary depth, and any number of inputs and outputs, as defined in (2).

To see what happens at initilization, we fix $\ell \geq 1$ and assume the network is initialized with i.i.d parameters satisfying $W^{(\kappa)} \sim \mathcal{N}(0, 1)$ and $b^{(\kappa)} \sim \mathcal{N}(0, 1)$. Then, in the (sequential) limit (when the number of all hidden neurons goes to infinity) $n_1, \dots, n_\ell \to \infty$, the output component functions $f^{(\ell+1)}_{\theta,\mu} : \mathbb{R}^{n_0} \to \mathbb{R}$, $\mu = 1, \dots, n_{\ell+1}$, converge in law to i.i.d. centered Gaussian processes $f^{(\ell+1)}_{\infty,\mu} : \mathbb{R}^{n_0} \to \mathbb{R}$ with covariance defined recursively by

$$\Sigma^{(1)}(x, y) = \frac{1}{\sqrt{n_0}} x^t y + 1 , \tag{14}$$

$$\Sigma^{(\kappa+1)}(x, y) = \mathbb{E}_{f\sim\mathcal{N}(0,\Sigma^{(\kappa)}(x,y))} [\phi(f(x))\phi(f(y))] + 1 . \tag{15}$$

In this context, the Neural Tangent Kernel (NTK) is the matrix valued Kernel whose components $\Theta^{(\ell+1)}_{\mu\nu} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ are defined by

$$\Theta^{(\ell+1)}_{\mu\nu}(x, y) = \sum_{\vartheta\in\theta} \frac{\partial f^{(\ell+1)}_{\theta,\mu}}{\partial\vartheta}(x) \frac{\partial f^{(\ell+1)}_{\theta,\nu}}{\partial\vartheta}(y) , \tag{16}$$

with $\theta = \{W^{(\kappa)}_{ij}, b^{(\kappa)}_k\}$ the set of all parameters. The fundamental observation by Jacot et al [11], discussed in the previous section, was in fact made for deep networks: if we initialize the parameters in a i.i.d fashion according to the law $\theta \sim \mathcal{N}(0, 1)$ then, as $n_1, \dots, n_{L-1} \to \infty$, the NTK converges, in law, to a deterministic kernel

$$\Theta^{(\ell+1)}_{\mu\nu} \to \Theta^{(\ell+1)}_{\infty,\mu\nu} = \Theta^{(\ell+1)}_\infty \delta_{\mu\nu} , \tag{17}$$

with the scalar kernel $\Theta^{(\ell+1)}_\infty : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ defined recursively by

$$\Theta^{(1)}_\infty(x, y) = \frac{1}{n_0} x^\intercal y + 1 , \tag{18}$$

$$\Theta^{(\kappa+1)}_\infty(x, y) = \Theta^{(\kappa)}_\infty(x, y)\, \dot\Sigma^{(\kappa+1)}(x, y) + \Sigma^{(\kappa+1)}(x, y) , \tag{19}$$

where, for $\kappa \geq 1$,

$$\dot\Sigma^{(\kappa+1)}(x, y) := \mathbb{E}_{f\sim\mathcal{N}(0,\Sigma^{(\kappa)}(x,y))} \left[\dot\phi(f(x))\, \dot\phi(f(y))\right] . \tag{20}$$

Similarly to what happens in the shallow case, for deep MLPs the constancy of the infinite width NTK $\Theta^{(\ell+1)}_\infty$, in parameter space, allows us to control the non-linear learning dynamics of large, but finite, width networks in terms of linear systems of ODEs [8, 12]. From this we can, once again, extract results concerning the memorization and generalization capacities of wide networks.

Moreover, in the deep network case, the eigenvalues of the NTK matrix $\Theta^{(\ell+1)}_\infty(X_i, X_j)$ are now func-

tions of the depth (and other hyper-parameters) and relevant notions of stability are linked to these quantities. For instance, it should be clear from (13) that $\lambda_{\min}$, the smallest eigenvalue of this matrix, controls the training convergence speed. So, if $\lambda_{\min}$ converges to zero, when the depth of the network goes to infinity, this means that the training will become arbitrarily slow with the increase of depth. On the other hand, if $\lambda_{\min}$ becomes arbitrarily large with the depth, then this means that the discretization of gradient descent requires, in practice, that we must keep on decreasing the learning rate (the quantity controlling the discretization size), when increasing the depth, to avoid large oscillations or divergences. In conclusion, when designing a network one would like to have $\lambda_{\min} \approx 1$, as a function of depth and other hyper-parameters, in order to have a robust learning setting. This criterium can be used to obtain some mathematical guidelines in fixing the hyper-parameters [6], in relation to question **Q4** raised in the introduction.

### 3.3 In conclusion: the two sides of the NTK

The NTK theory has opened a window into the learning dynamics of wide neural networks [9], that led to remarkable progress concerning the theoretical understanding of their performance. Nonetheless, the fact that learning with sufficiently wide MLPs is, in essence, restricted to kernel regression is, in a sense, even more disappointing than the Gaussian behavior of MLPs at initialization. To justify this emotional reaction, consider that entering a kernel regime implies, in particular, that all features are fixed at initialization and learning becomes restricted to finding appropriate linear combinations of these features. This situation is sometimes referred to as *lazy training*. This inability to perform data-driven *feature learning* has other unfortunate consequences: for instance, it prevents the use of powerful techniques such as *transfer learning*, where features learned in a "big" data set are later reused in fitting a new model to a smaller dataset [10]. Such results show that simply scaling up the number of units per layer can lead to undesirable scenarios, which relates back to **Q3** raised in the introduction. Nonetheless, several proposals [5, 19] have been set forth to circumvent these difficulties and retain *feature learning*, even in the large width regime.

## 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*, In Advances in Neural Information Processing Systems, 32, 2018.

[2] M. Belkin, S. Ma, and S. Mandal, *To understand deep learning we need to understand kernel learning.* International conference on machine learning. PMLR, 2018.

[3] L. Carvalho, J. L. Costa, J. Mourão, and G. Oliveira, *Wide neural networks: From non-Gaussian random fields at initialization to the NTK geometry of training*, https://arxiv.org/abs/2304.03385

[4] L. Carvalho, J. L. Costa, J. Mourão, and G. Oliveira, *The positivity of the neural tangent kernel*, SIAM journal of Mathematics of Data Science, Volume 7, Issue 2 , 2025

[5] L. Carvalho, J. L. Costa, J. Mourão, and G. Oliveira, *A ZeNN architecture to avoid the Gaussian trap*, https://arxiv.org/pdf/2505.20553.

[6] L. Carvalho, J. L. Costa, J. Mourão, and G. Oliveira, *The spectrum of the NTK and the stability of training.*, in preparation.

[7] F. Chollet, *Deep Learning with Python, Second Edition*, Manning Publications Co (2021).

[8] S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, *Gradient descent finds global minima of deep neural networks*, International conference on machine learning, 2019.

[9] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press 2019, http://www.deeplearningbook.org.

[10] J. Hron, Y. Bahri, J. Sohl-Dickstein, and R Novak *Infinite attention: NNGP and NTK for deep attention networks.* International Conference on Machine Learning. PMLR, 2020.

[11] A. Jacot, G. Franck, and C. Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, Advances in Neural Information Processing Systems (2018), 8580–8589.


[9] Here we have only discussed the MLP case, but the NTK theory extends to other architectures, see for instance [10].
[10] Recall that large models are prone to overfitting if the dataset is small.

[12] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, *Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent*, Advances in Neural Information Processing Systems 32 (NeurIPS 2019)

[13] Y. Li, H. Zhang, and Qian Lin *Kernel interpolation generalizes poorly* Biometrika, Volume 111, Issue 2, 2024

[14] W.S McCulloch, and W. A. Pitts, *logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259

[15] N. Radford, *Priors for infinite networks*, tech. rep. no. crg-tr-94-1, University of Toronto, 1994.

[16] F. Rosenblatt, *The Perceptron – a perceiving and recognizing automaton*, Report 85-460-1. Cornell Aeronautical Laboratory.

[17] M. Telgarsky, *Representation benefits of deep feedforward networks*, https://arxiv.org/abs/1509.08101

[18] A. Vaswani , N. Shazeer, N. Parmar, J Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin *Attention is all you need*, Advances in neural information processing systems 30 (2017).

[19] G. Yang, and E. J. Hu, *Tensor programs iv: Feature learning in infinite-width neural networks*, International Conference on Machine Learning, 2021.

[20] M. Zeff *Current AI scaling laws are showing diminishing returns, forcing AI labs to change course* TechCrunch, https://techcrunch.com/2024/11/20/ai-scaling-laws-are-showing-diminishing-returns-forcing-ai-labs-to-change-course/?utm_source=chatgpt.com