UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

TWO CONTRIBUTIONS TO PROBABILISTIC ML: BAYESIAN SPECTRAL ESTIMATION & EXTENSION TO THE NTK

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MATEMÁTICAS APLICADAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

SEBASTIÁN ANIBAL LÓPEZ TORO

PROFESOR GUÍA:
FELIPE TOBAR HENRÍQUEZ

MIEMBROS DE LA COMISIÓN:
JOAQUÍN FONTBONA TORRES
MAURICIO ARAYA LÓPEZ
JORGE SILVA SÁNCHEZ

Este trabajo ha sido parcialmente financiado por:
CMM ANID PIA AFB170001 y Fondecyt-Iniciación 11171165

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA,
MENCIÓN MATEMÁTICAS APLICADAS
RESUMEN DE LA MEMORIA PARA OPTAR
AL TITULO DE INGENIERO CIVIL MATEMÁTICO
POR: SEBASTIÁN ANIBAL LÓPEZ TORO
FECHA: 2023
PROF. GUÍA: FELIPE TOBAR HENRÍQUEZ

DOS CONTRIBUCIONES AL ML PROBABILÍSTICO: ESTIMACIÓN ESPECTRAL BAYESIANA & EXTENSIÓN DEL NTK

En las áreas de procesamiento de datos y aprendizaje de máquinas, dos ramas importantes son procesamiento de señales y aprendizaje profundo, respectivamente. Procesamiento de señales se centra en el análisis, modificación y sintetización de señales. Por otra parte, aprendizaje profundo es un conjunto de métodos basados en el uso de redes neuronales multicapa.

Una de las áreas más antiguas en procesamiento de señales es la estimación espectral (SE), la cuál estima la densidad espectral (PSD) de una señal a partir de un conjunto de observaciones ruidosas de la señal. Uno de los métodos clásicos es la estimación espectral Autoregresiva (ASE), la cuál impone que la señal estudiada proviene de la familia de procesos Autoregresivos, y luego encuentra el proceso Autoregresivo que más se le asemeja mediante técnicas de optimización. Comunmente, los procecsos usados son máxima verosimilitud o mínimos cuadrados, por lo que esta estimación no toma en cuenta la incertidumbre derivada del hecho de ajustar una señal completa a partir de un conjunto finito de observaciones.

Hace unos años, Arthur Jacot probó que las redes neuronales Bayesianas entrenadas con descenso del gradiente tenían una correspondencia con los procesos Gaussianos, y que los cambios experimentados por la red durante el procecso de entrenamiento estaban fuertemente ligados al kernel de este proceso Gaussiano[1]. A pesar de los grandes avances que este campo ha visto, ha existido poca investigación que utilize otras configuraciones, dado que la mayoría de trabajos usan redes neuronales profundas entrenadas con descenso del gradiente y utilizan el error cuadrático medio como función de costo.

Para tomar en cuenta la incertidumbre derivada del proceso de ajuste en la estimación espectral Autoregresiva, proponemos un acercamiento Bayesiano al problema, usando una distribución a priori en los parámetros del proceso Autoregresivo, así podemos obtener en última instancia una distribución posterior en la densidad espectral de la señal, y así obtener una medida de la incertidumbre de nuestra predicción simplemente mirando la varianza de la distribución. Para expandir la investigación en redes neuronales Bayesianas y su correspondencia con procesos Gaussianos, probaremos que esta correspondencia se mantiene para redes neuronales entrenadas utilizando la entropía cruzada como función de costo, la cuál es común en problemas de clasificación. Esto se hará primero, probando que la entropia cruzada tiene todas las propiedades requeridas para que exista la correspondencia, y luego comprobandolo con experimentos apropiados.

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA,
MENCIÓN MATEMÁTICAS APLICADAS
RESUMEN DE LA MEMORIA PARA OPTAR
AL TITULO DE INGENIERO CIVIL MATEMÁTICO
POR: SEBASTIÁN ANIBAL LÓPEZ TORO
FECHA: 2023
PROF. GUÍA: FELIPE TOBAR HENRÍQUEZ

TWO CONTRIBUTIONS TO PROBABILISTIC ML: BAYESIAN SPECTRAL ESTIMATION & EXTENSION TO THE NTK

In the data processing and machine learning field, two important branches are signal processing and deep learning, respectively. Signal processing focuses on analysing, modifying and synthesizing signals. On the other hand, deep learning is a set of methods that are based on the use of neural networks with multiple layers.

One of the oldest fields in signal processing is spectral estimation (SE), which is concerned with estimating the Power spectral density (PSD) of a signal from a set of noisy observations of this signal. One of the classic methods is Autoregressive spectral estimation (ASE), which imposes that the signal belongs to the family of Autoregressive processes, and then finds the best fitting autoregressive process through an optimization procedure. Commonly, the procedures used are maximum likelihood or least squares, so this method does not take into account the uncertainty derived from fitting an entire signal only from a finite set of observations.

In recent years, Arthur Jacot proved that Bayesian networks trained with gradient descent have a correspondence with Gaussian Process, and that the changes in the network during the training procedure were deeply related to the kernel of this Gaussian process[1]. Nevertheless, despite the great advances this field has seen, there has been little research in using other configurations, since most of the research uses deep neural networks trained with gradient descent and mean squared error cost.

To take into account the uncertainty derived from the fitting process in the Autoregressive spectral estimation, we propose a Bayesian approach to this problem, by using a prior distribution on the parameters of the Autoregressive process, we can ultimately obtain a posterior distribution on the Power spectral density of the signal. Since we have a distribution, we can easily obtain a measure of the uncertainty of our prediction by looking at the variation of this distribution. To extend the research on Bayesian deep networks and their correspondence with Gaussian processes, we will prove that this correspondence holds for networks trained using the cross entropy cost, usually used in classification problems. This will be done first by proving that the cross entropy cost has all the required properties for the correspondence to exist, and then by testing it out with appropriate experiments.

“Alea iacta est”

— Julio César

Agradecimientos

Agradezco a mi familia, quien me ha apoyado en cada una de mis decisiones académicas, alentándome a estudiar, aprender y a desafiarme cada vez que fuera posible, pero también me dieron la libertad y la posibilidad de distenderme y conocerme en otros aspectos de mi vida. Agradezco en especial a mi madre, Natalia, quien me inculcó sentido de la responsabilidad y deber, los cuáles fueron vitales a la hora de elegir este camino.

Agradezco a mis amigos del colegio, los HC, con quienes conocí muchas cosas propias de la juventud, y con quienes viví muchos momentos críticos que me formaron como persona. Nunca olvidaré las tallas de los momentos de distensión, y las reflexiones que tuvimos en los momentos serios. Agradezco en especial a Marquito, quién me conoce más que nadie, y ha estado ahi para escucharme cada vez que lo he necesitado.

Agradezco a mis amigos de plan común, los 3R, con quienes descubrí la vida universitaria. Gracias a ellos puedo decir que esta etapa deja más que solo conocimientos, también deja personas valiosas y variadas, que aportan de diversas maneras en mi vida, con sus puntos de vista y enfoques distintos.

Agradezco a mis amigos de especialidad, los Chacritas, con quienes atravesamos en conjunto uno de los peores momentos de la universidad. Pero en la adversidad se forjan las buenas amistades, y su apoyo en los momentos más duros, su constante ayuda en el estudio, y su apañe en los momentos de distención, cada Bangazo, cada oncecita, cada papita, cada shawarma y cada chelita, me mantuvieron con el ánimo y la energía necesarias para enfrentar cada día de la especialidad. Sin ustedes hubiera sido imposible terminar esta carrera, y por eso siempre les estaré agradecido.

También agradezco a Gillian, quien me brindó su amor y cariño durante todos mis años universitarios. Celebró conmigo en mis grandes alegrías, pero también me consoló cuando parecía que todo estaba perdido. Me dió su comprensión, y me brindó muchos años de felicidad. Siempre estaré agradecido de tu paso por mi vida.

Agradezco a mis amigos de la vida, Paz y Gabriel, quienes me han brindado grandes conversaciones, para desahogarme o entenderme a mi mismo. Siempre un hombro donde apoyarme, o un oído a quien confiarle mis asuntos más íntimos, de manera comprensiva y nunca juzgándome. Por eso les agradezco mucho, y espero que cuenten conmigo en sus vidas.

A todos quienes mencioné aquí, tienen un lugar especial en mi corazón, y parte de quien soy hoy es gracias a ustedes, los llevaré siempre conmigo.

Table of Content

Chapter 1

What is Machine Learning and why it is important.

Machine Learning (ML) is a branch of Artificial Intelligence (AI) and Computer Science (CS), devoted to the theory, understanding, construction and implementation of methods and algorithms that can find patterns among historical data, and "learn" from it, in order to improve its performance on a given set of tasks.

The most prominent characteristic of this methods and algorithms, is that they are not explicitly programmed to complete their tasks in a certain explicit or specific way, but rather we let the algorithm itself discover the best approach to complete such task, by identifying any correlation, similarity, patterns or important features present in the given data, and use the collected information in the future, when trying to complete the task in new and unseen data.

The term "Machine Learning" was first used in 1959, by an IBM employee named Arthur Samuel, who was a pioneer in the field of artificial intelligence, specifically in computer gaming. Since then, the methods, algorithms, techniques and objectives of the machine learning field have evolved dramatically, thanks to the interest that the scientific community has put in developing this field, specially in the most recent years, but also thanks to the incredible increase on computational power, which was a major barrier on how much the algorithms could learn from data during the last century.

These methods and algorithms are used in a wide variety of tasks, such as classification, regression, clustering, anomaly detection, association finding, dimensionality reduction, machine translation, fraud detection, data labelling, robotics, gaming bots, resource management, image segmentation, speech recognition and computer vision, among many others tasks. Most importantly, solving these tasks in an efficient way is important in many areas and applications such as medical diagnosis, tumor segmentation, recommendation engine, self-driving vehicles, virtual assistants, business intelligence, human resource management, and many others that impact the daily routines of people.

Machine learning has become a corner stone in many areas because it brought a huge improvement in many of the tasks previously mentioned. For example, using image processing algorithms, we can improve the image resolution of medical resonances or X-ray, or automatically detect cellular anomalies during an exam, thus making it easier and faster for the medical teams to diagnose diseases and treat them accordingly. This is why many of the medical equipment nowadays, such as MRI and X-ray scanners, include in their software machine learning algorithms to improve the quality of the images. Recommendation engines, such as the ones used by Youtube and Netflix, analyze the patterns of an user, in order to give more accurate recommendations, suited for each user likings. This have been revolutionary in business models, which now can be customized for each individual user, instead of making general business models aimed for generic group of costumers.

Machine learning, as a field, has many sub-fields, which focus on different and specific tasks. For example, two prominent areas are **Signal Processing** and **Deep Learning**.

Signal processing refers to the methods and algorithms that are used for analysing, modifying and synthesizing signals. These signals can come from various sources, such as music, speech, medical exams, electronic components, telluric movements, and many other. The techniques used focus on optimizing the acquisition and transmission of these signals, to make the storage of the signals more space efficient, to correct distortion on such signals, or to detect any component of interest or anomaly in the signals. This is one of the first and classical sub-fields, which existed from before, but received a new and renovated approach when the Machine learning techniques appeared.

On the other hand, Deep learning is a relatively new sub-field compared to Signal processing, which refer to algorithms, typically artificial neural networks, that posses multiple layers to extract information and process the data, in order to learn how to represent more complex functions, that other kind of algorithms cannot learn due to the simplicity of their architectures. For example, when processing an image, the first layers of a deep neural network tend to focus on the fine details of the mages, such as edges or borders, meanwhile the latter layers tend to focus on more general concepts of the image, such as recognizing what the image represents: an animal, a digit, a letter, a face, among others concepts.

In this thesis, we will make advances in each of the previously mentioned sub-fields. In Signal processing, we will take a classic but obsolete spectral estimation method, called the Auto-regressive spectral estimation (ASE), and give it a renovated, fresh approach by adding a Bayesian treatment to this method, thus proposing a new method called Bayesian Auto-regressive spectral estimation (BASE). We then analyze the properties and limitations of this new method.

In Deep learning, we will make progress in understanding the theoretical process that is underlying when using deep neural networks. Simply put, we will investigate how the network actually learns when trained on data. For this purpose, we will use the Neural Tangent Kernel approach, and extend it to a different cost function.

We expect this work to be useful to the scientific community, and specifically to the Machine learning community, and that it will be another small step on the march of progress, by helping us to better understand the algorithms and methods we use, as well to generate new ideas and applications of such algorithms.

Chapter 2

Bayesian Autoregressive Spectral Estimation

2.1. Introduction

In Signal Processing, spectral estimation (SE) refers to the determination of the energy that is contributed by each component in a time series. In particular, we will focus on oscillatory periodic single-frequency components, the spectral representation of which is represented by the Fourier power spectral density (PSD). In practice, the challenge is to estimate the PSD from a set of noisy observations of the time series; this is seen in applications such as biomedical engineering [2] and telecommunications [3]. Classical methods for computing the PSD can be roughly divided in two categories, parametric and nonparametric. Parametric methods impose a generative model on the data such as autoregressive moving average (ARMA) models [4], sums of sinusoids (Lomb-Scargle [5]), Bernstein polynomials [6] or Gaussian mixtures [7]. On the other hand, nonparametric methods for spectral estimation do not assume any structure on the data and are thus more flexible, yet they do not discriminate the signal from the noise, which critically affects the spectral estimates. Typical examples of the nonparametric SE methods are the classic Periodogram [8] which can be considered as a *histogram of frequencies* as well as recent Bayesian nonparametric approaches [9, 10, 11].

Even though the randomness conveyed by time series observations is evident, dealing with the uncertainty related to the data-driven computation of the PSD is not a standard practice within SE. In parametric approaches, the model is imposed on the data through an optimisation procedure, e.g. in the least squares or maximum likelihood senses. Therefore, the PSD is approximated deterministically, by a point estimate, and all the information about the mismatch between the chosen model and the data, which could lead to probabilistic PSD estimates, is unfortunately usually neglected.

To address this scenario, which is particularly detrimental in noisy data and model mismatch, we aim to cater for uncertainty in PDS estimates finding a *distribution over PSDs* rather than point estimates. This can be achieved by equipping the existing framework for parametric spectral estimation with a Bayesian treatment. Instead of committing to a point-estimate of parameters of the time-series model, we can find their posterior distribution, thus constructing a posterior distribution over models for time series. Then, we can *transport* this distribution via the Fourier transform to find the posterior distribution over PSDs, where *posterior* in this case is in the sense of conditional to the time series observations. This rationale becomes of particular importance for model misspecification, since, strictly speaking, a parametric model may always be considered to be misspecified for real data and therefore the distribution of models allows us to account for the limited capacity of the parametric representation.

In this paper, we apply this generic idea to the particular class of autoregressive time series models, thus our proposed method is referred to as Bayesian autoregressive spectral estimation (BASE). In particular, we compute the posterior distributions over the AR parameters via Markov chain Monte Carlo (MCMC) [12], then, we use the MCMC samples to produce a sample version of the PSD, which is closed-form given the AR parameters. Via simulations, we validate the proposed BASE method on two synthetic signals and a real world example, these experiments show the advantage of the proposed BASE model in terms of robustness to order misspecification, unbiasedness, uncertainty representation and periodicity detection.

Also, in an effort to avoid the heavier part of the computational cost, derived from the use of the MCMC algorithm, we propose an special prior distribution, which due to the nature of the AR process, is conjugate with the posterior distribution. This allow us to directly compute the parameters by simply extracting a sample from the posterior distribution, which is explicit upon defining the prior distribution.

The work presented in this section was done jointly with Alejandro Cuevas, Danilo Mandic and Felipe Tobar. It was published and presented at the 2021 IEEE Latin American Conference on Computational Intelligence (LA-CCI)[13]

2.2. Background: Parametric spectral estimation using an AR model

A discrete-time sequence $\{x_t\}_{t\in\mathbb{N}}$ is called a p-order autoregressive process, denoted AR(p), if it is given by the relationship [8]

$$x_t = \sum_{k=1}^{p} a_k x_{t-k} + \epsilon_t, \tag{2.1}$$

where $p \in \mathbb{N}$, $[a_1, \dots, a_p] \in \mathbb{R}^p$ are the (autoregressive) parameters and $\{\epsilon_t\}_{t\in\mathbb{N}}$ is a noise process. We consider the noise to be Gaussian independent and identically distributed random variables with zero mean and variance σ^2.

2.2.1. Power spectral density of AR(p) processes

AR(p) processes are ubiquitous in parametric spectral estimation as they have closed-form PSDs [14] given explicitly by the AR parameters. Specifically, using the Z-transform [4] the PSD of AR(p) can be expressed by

$$S_x(\xi) = \frac{\sigma^2}{|1 - \sum_{k=1}^{p} a_k e^{-i2\pi\xi k}|^2}. \tag{2.2}$$

The modes (or peaks) of the PSD in eq. (2.2), i.e., the frequencies where the AR process convey more power, are given by the roots of the denominator. These roots are the poles of the dynamical system that generates the AR(p) process $\{x_t\}_{t\in\mathbb{N}}$. Since we can regard the model order p as the degrees of freedom of the parametric PSD for the AR model, the larger the p the more flexible the PSD; this is in line with the interpretation in the temporal domain, where a model with more lags can cater for more complex time series and thus represents a more general model.

The fact that the PSD above is directly parametrised by $\{a_1, \ldots, a_p, \sigma_n\}$ is perhaps the main motivation for using the AR model for spectral estimation. However, even though there have been Bayesian approaches to fit the AR parameters they have not been applied to the SE problem. In practice, the literature shows that the use of AR parameters in SE has been mainly deterministic, e.g., by computing the PSD in eq. (2.2) with a plug-in approximation of the AR parameters such as those found by least squares, the method of moments, or maximum likelihood. Such an approach, does not allow for modelling the uncertainty related to the peaks of the resulting PSD, due to noisy data, or to account for model misspecification (i.e., incorrect model order) which can lead to either peak splitting (over-estimation) or onto fewer peaks (under-estimation).

2.3. Bayesian spectral estimation for the autoregressive model

Our aim is to compute the posterior distribution of the PSD for the AR case, thus providing a natural account for model uncertainty, which arises from i) the consideration of a finite dataset, ii) model mismatch, iii) noisy observations. From now on, we will refer to the standard (deterministic) autoregressive spectral estimation approach as ASE and to the proposed Bayesian counterpart as BASE.

2.3.1. Generic Bayesian parametric spectral estimation

The posterior distribution of the PSD of a stochastic process $\{x\}_{t\in\mathbb{N}}$, denoted by S_x, conditional to a set of observations $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, can be computed by integrating out the model of the time series. Denoting the space of all possible time series models by , the posterior PSD is given by

$$\begin{aligned} p(S_x|\mathbf{x}) &= \int_p (S_x, M|\mathbf{x}) dM \\ &= \int_p (S_x|M) p(M|\mathbf{x}) dM, \end{aligned} \tag{2.3}$$

where the last expression only takes the (reasonable) assumption that the PSD, S_x, and the observations, $\mathbf{x}$, are conditionally independent given the model M. This follows from the fact that the data cannot say more about the PSD than the model itself.

Notice that the above integral is defined over the entire, possibly infinite-dimensional, model space . To calculate this integral we need to assume a structure for the time-series models to be considered; this can be achieved by choosing a finite-dimensional prior over the models, namely $p(M)$. A straightforward way to choose this prior is to first assume a parametrisation over models, say $M = M_\theta,\ \theta \in \mathbb{R}^d$, and then choose a prior over the parameters $p(\theta)$. This construction is know as the push-forward measure [15] that transports a distribution over the finite-parameter, $\theta \in \mathbb{R}^d$, towards the space of models through the mapping $\theta \to M_\theta$.

With this parametrisation, the posterior in eq. (2.3) can be expressed with respect to the model parameters, θ, via the change of variable theorem as

$$p(S_x|\mathbf{x}) = \int_\Theta p(S_x|\theta)p(\theta|\mathbf{x})d\theta, \tag{2.4}$$

where the integration is now performed over the (finite) parameter space, that is, $\Theta = \mathbb{R}^d$.

Under the assumption that the process $\{x_t\}_{t\in\mathbb{N}}$ is stationary and given by a model M_θ, its PSD is uniquely defined by the model's parameters θ and thus the distribution $p(S_x|\theta)$ is a Dirac measure supported on a single points on the space of PSDs. Therefore, sampling from the posterior over PSDs, $p(S_x|\mathbf{x})$, is straightforward: simply sample a parameter from the posterior distribution over parameters $\theta^* \sim p(\theta|\mathbf{x})$, and then map this parameter sample to PSDs according to

$$\theta^* \to M_{\theta^*} \to S_x = \mathrm{PSD}(M_{\theta^*}), \tag{2.5}$$

where $\mathrm{PSD}(M)$ denotes the PSD corresponding to the model M.

In order to perform this procedure, we need to choose a model space M_θ, a prior over the parameters $p(\theta)$ and a sampling procedure. We refer to these in the remaining part of this section.

2.3.2. Establishing a prior over AR models

We shall choose the model space as that of AR models owing to their appealing properties for spectral estimation outlined in Section 2.2.1. Furthermore, we will assume that all parameters in the AR model are independent, that $a_{1:p}$, $p \in \mathbb{N}$, are Normally distributed and the noise variance σ^2 follows a half-Normal prior. Thus, the priors are

$$p(a_{1:p}) = (0, \sigma_{ap}^2) \tag{2.6}$$
$$p(\sigma^2) = \text{half-}(0, \sigma_\epsilon^2), \tag{2.7}$$

where the half-Normal distribution corresponds to the multiplication between a Normal distribution and an indicator function $_{\{\sigma^2\geq 0\}}$, and then properly renormalised (i.e., multiplied by 2).

The choice of normality for the autoregressive parameters follows from the fact that the Gaussian prior is conjugate to the (conditional) likelihood as verified in the next section, and thus the posterior is also normally distributed.

2.3.3. Model likelihood

For a given set of observations $\{x_{1:T}\}$ we focus on the **conditional** likelihood of the AR(p), that is, the expression $p(x_{p+1:T}|x_{1:p},\theta)$, where $\theta = [a_1,\dots,a_p,\sigma^2]^\top$ denotes all model parameters and we consider $\{x_{1:p}\}$ to be fixed and not part of the generative model for simplicity of presentation.

Following eq. (2.1), the conditional likelihood of the AR(p) model is given by

$$\begin{aligned} p(x_{p+1:T}|x_{1:p},\theta) &= \prod_{\tau=p+1}^{T} p(x_\tau|x_{\tau-1:\tau-p},\theta) \\ &= \prod_{\tau=p+1}^{T} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x_\tau -^\top \mathbf{x}_{\tau-1})^2}{2\sigma^2}\right), \end{aligned} \tag{2.8}$$

where we have adopted the notation $= [a_1,\dots,a_p]$ and $\mathbf{x}_{\tau-1} = [x_{\tau-1},\dots,x_{\tau-p}]$.

The conditional likelihood is thus Gaussian on the AR parameter vector , meaning that the Gaussian prior on established on the previous section is conjugate to the conditional likelihood and results on the following Gaussian posterior

$$\begin{aligned} p(\theta|x_{1:p},x_{p+1:T}) &= \frac{p(x_{p+1:T}|x_{1:p},\theta)p(\theta)}{p(x_{p+1:T},x_{1:p})} \\ &\propto p(x_{p+1:T}|x_{1:p},\theta)p()p(\sigma^2) \\ &\propto \prod_{\tau=p+1}^{T} \frac{1}{\sigma} \exp\left(\frac{-(x_\tau -^\top \mathbf{x}_{\tau-1})^2}{2\sigma^2}\right) \\ &\quad \cdot \exp\left(\frac{-1}{2\sigma_a^2} a^T a\right) \exp\left(\frac{-\sigma^2}{2\sigma_\epsilon^2}\right). \end{aligned} \tag{2.9}$$

The complete (rather than conditional) likelihood is given by

$$p(x_{1:T}|,\theta) = p(x_{p+1:T}|x_{1:p},\theta)p(x_{1:p}|\theta), \tag{2.10}$$

and thus implies computing the distribution $p(x_{1:p}|\theta)$. This quantity is, however, difficult to calculate since it involves the infinite-order moving-average representation of the AR(p) process. Therefore, under the assumption that the first p observations are fixed they become independent from θ, and the density $p(x_{1:p}|\theta)$ becomes a constant for theta in eq. (2.10). A similar reasoning applies for the marginal distribution $p(x_{p+1:T},x_{1:p})$. Notice that the discrepancy between the conditional and complete likelihoods becomes negligible for increasing amounts of data. For this reason, we will consider only the conditional likelihood.

With this choice, generating a sample, θ^*, from the posterior requires the evaluation of $p(x_{p+1:T}|x_{1:p},\theta)p()p(\sigma^2)$. Then, this sample can be used to compute the PSD explicitly, according to eq. (2.2). This procedure avoids explicit computation of the model M_θ, since it allows us to compute the PSD directly from the parameters.

Notice that, by construction, all the posterior PSD samples will have the form as in eq. (2.2) which means that no probability will be assigned to expressions that do not follow this form. Therefore, using a set of samples of PSDs constructed as explained above, we can numerically compute the mean PSD and error bars conditional to a set of signal values $\mathbf{x}$.

2.3.4. Closed form posterior

If we assume instead that the noise variance σ^2 follows an Inverse-Gamma distribution, we obtain a closed expression for the posterior distribution on the parameters, and it is a Normal-Inverse-Gamma distribution. More specifically, let the priors be:

$$p(a_{1:p}|\sigma^2) = \left(\mu_0, \frac{\sigma^2}{\lambda}_p\right) \tag{2.11}$$

$$p(\sigma^2) = \Gamma^{-1}(\alpha, \beta) \tag{2.12}$$

Then the posterior distribution for $\theta = (a_{1:p}, \sigma^2)$ is proportional to a Normal-Inverse-Gamma:

$$p(\theta|x_{1:T}) \propto \Gamma^{-1}(\mu, \Sigma^{-1}, \overline{\alpha}, \overline{\beta}) \tag{2.13}$$

where:

$$\begin{aligned} X_{p+1:T} &= [x_{i-1:i-p}^\top]_{i=p+1}^T \\ \Sigma^{-1} &= \lambda_p + X_{p+1:T}^\top X_{p+1:T} \\ \mu &= \Sigma(\lambda\mu_0 + X_{p+1:T}^\top x_{p+1:T}) \\ \overline{\alpha} &= \alpha + \frac{T-p}{2} \\ \overline{\beta} &= \beta + \frac{\lambda\mu_0^\top\mu_0 + x_{p+1:T}^\top x_{p+1:T} - \mu^\top\Sigma^{-1}\mu}{2} \end{aligned}$$

Using this closed-form distribution, we can obtain a sample of the parameters $(a_{1:p}, \sigma^2)$ simply by specifying the hyper-parameters $(\mu_0, \lambda, \alpha, \beta)$ and sampling from the distribution, allowing us to avoid the process of MCMC, which can be slow in certain cases.

To get better estimations, we made an hyper-parameter tuning using a Grid search and a cross-validation scheme with a 5-fold Time series split. The scheme consist of the following: For each combination of hyper-parameters in the grid, we compute the Maximum a posteriori estimator (MAP) on the train split of the fold, to then score it on the test split of the fold using the likelihood of the data given the MAP for $a_{1:p}$ and σ^2 as the scorer function.

The MAP estimators for the parameters $(a_{1:p}, \sigma^2)$, asuming we have m observations of the signal, are given by

$$\begin{aligned} a_M &= \left(\lambda_p + \sum_{i=1}^m X^\top X\right)^{-1}\left(\lambda\mu_0 + \sum_{i=1}^m X^\top x\right) \\ \sigma_M^2 &= \frac{2\beta + \lambda(a_M-\mu_0)^\top(a_M-\mu_0) + \sum_{i=1}^m (x - Xa_M)^\top(x - Xa_M)}{2(\alpha+1) + mT + (m-1)p} \end{aligned}$$

where $X := X_{p+1:T}$ and $x := x_{p+1:T}$. The final score assigned to the hyper-parameters is the average of the scores obtained on each fold. The set of hyper-parameters with the best score will be used to compute the posterior distribution of the parameters.

The grid used for the hyper-parameters are

$$\lambda, \alpha, \beta \in \{0.1, 1, 10, 100\}$$
$$\mu_0 \in \left\{ \begin{bmatrix} -10 \\ -10 \\ \vdots \\ -10 \end{bmatrix}, \begin{bmatrix} -8 \\ -8 \\ \vdots \\ -8 \end{bmatrix}, \dots, \begin{bmatrix} 10 \\ 10 \\ \vdots \\ 10 \end{bmatrix} \right\} \subset \mathbb{R}^p$$

2.4. Simulations

We validated the proposed Bayesian autoregressive spectral estimation method, BASE, over three case studies, two for synthetic data and one for real world data. The first one considered a synthetic AR signal, where the true and approximate model orders were different, this simulation is aimed to show that BASE is robust to model misspecification due to the chosen prior. The second experiment illuminates the unbiasedness and concentration properties of the approximate posterior over PSDs by applying BASE on a continuous-time signal generated by a Gaussian process (GP) [16] with known PSD. Finally, the third experiment validates the ability of the proposed BASE model to detect periodicities from a sub-sampled real world astronomical time series. We also compare, over all the case studies, the performance of the method in both versions: using MCMC and closed-form. In all experiments, we sampled from the posterior using MCMC, specifically, we used a NUTS sampler [17] in the PyMC3 Python toolbox [18].

2.4.1. Misspecified model order: AR-generated time series

We implemented a stable AR(4) time-series to generate data that was later processed by BASE with the assumption of a AR(10) model. Fig. 2.1 shows the true values of the regression coefficients and their approximate posterior using MCMC; notice that these posteriors contain the true AR(4) coefficients, and they also converge to narrow posteriors centred around zero for higher orders coefficients. This validates the self-regularisation property, or robustness to misspecified model order, within the proposed BASE-framework which will be key when computing the sample PSDs.

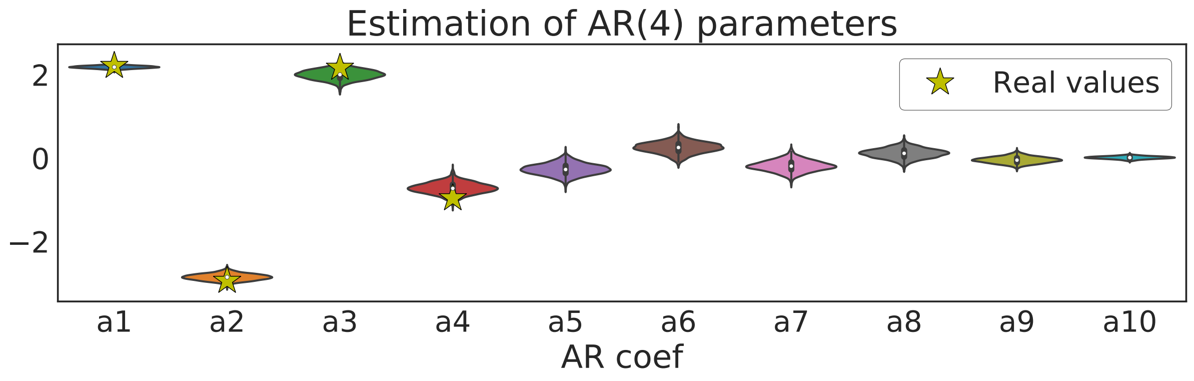

Figure 2.1: Approximate posterior distributions over AR parameters using MCMC (NUTS) and the true AR(4) values shown in yellow stars.

Next, using the approximate posterior over the AR parameters (coefficients and variance), we generated PSD samples according to eq. (2.2). Fig. 2.2 shows the density over PSDs using the proposed BASE (95% error bars), the standard Autoregressive spectral estimation, the true PSD and the Periodogram. Notice that the proposed BASE succeeded in estimating the true PSD while remaining unbiased and concentrated around the true value.

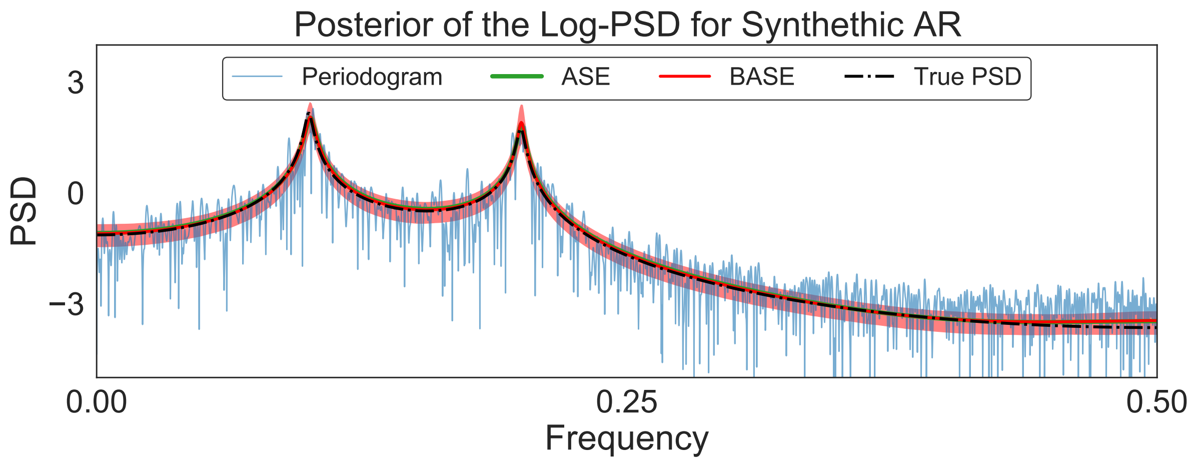

Figure 2.2: Log-PSD estimate for a synthetic AR(4) signal using the BASE method; True PSD (dashed black line), Periodogram (blue line), ASE (green line), and proposed BASE (red line, 95% error bars).

2.4.2. Continuous-time signal: Gaussian process with Laplace covariance

A synthetic signal was generated from a Gaussian process (GP) [16] with Laplace covariance function given by

$$K(\tau) = \sigma^2 \exp(-|\tau|/l), \tag{2.14}$$

where σ^2 denotes the marginal covariance and l is known as the process *lenghtscale*. Furthermore, we considered zero-mean Gaussian noise added to the GP sample.

Recall that a sample from a GP can be regarded as a signal generated by an AR(∞) model. Therefore, the proposed BASE model was implemented with an AR(4) model, the order of which was chosen from the rate of decay of the Laplace kernel. Then, the PSD estimate of the BASE model was compared against those of the standard ASE of the same order and the Periodogram. Fig. 2.3 shows the PSD estimates (95% error bars for BASE) as well as the true PSD of the signal, given by the Fourier Transform of the Laplace kernel. Notice that the PSD posterior within BASE was concentrated around the true PSD and was able to capture the mean of its deterministic counterpart.

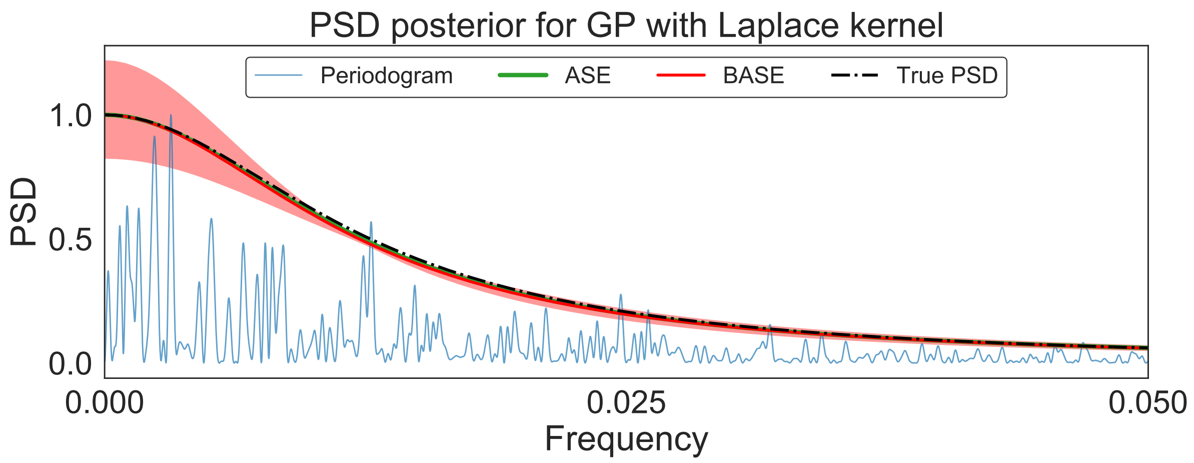

Figure 2.3: Spectral estimation of a GP trajectory: BASE shown in red, true PSD in dashed black, Periodogram in blue and ASE in green. The figure only shows the region [0, 0.05], since this is where almost all the spectral content of the data is contained.

2.4.3. Finding the periodicity of an astronomical time series

For the third experiment, we considered a real-world time series which may not be an AR process and therefore its PSD will not be precisely given by the expression in eq. (2.2). This experiment was constructed to validate the Bayesian approach to handle model misspecification by averaging over the posterior models. The signal considered was the sunspots dataset [19], known for having a period of 11 years (frequency ≈ 0.0909[years^{-1}]), we implemented BASE to detect this periodicity only using 1/6 of the available data.

We first computed both the autocorrelation (ACF) and partial autocorrelation (PCF) functions of the sunspots series. Fig. 2.4 shows the ACF and PCF, and show that due to the slow decay of PCF, the order of the autoregressive component of the signal cannot be specified with any certainty; this supports the evidence that the sunspot series is not an AR process. Nevertheless, we proceeded by choosing an AR(9) model and implemented BASE to compare against the ASE and the Periodogram, where ASE and the Preiodogram used all the available data. Fig. 2.5 shows the peak estimates of the PSDs using BASE, where the main peak (colour-coded in red) is 0.09181278[years^{-1}] and thus in line with the expected frequency peak. Fig. 2.6 shows the PSD estimates for the three models, with the proposed BASE model exhibiting its peak near the correct known period of the sunspots signal, as it was expected from Fig. 2.5.

Notice that the errors bars of BASE are now larger than those of the previous experiments, this is because the signal considered in this experiment is not coming from an AR process and thus we are in the scenario of model mismatch. In this sense, due to the fact that BASE averages over possible AR models, its error bars are larger but they contain other peaks of the Periodogram. Finally, observe that despite the fact that an order $p = 9$ for the model was chosen, this did not result in multiple peaks which validetes BASE's robustness to overtiffing arising from its Bayesian nature.

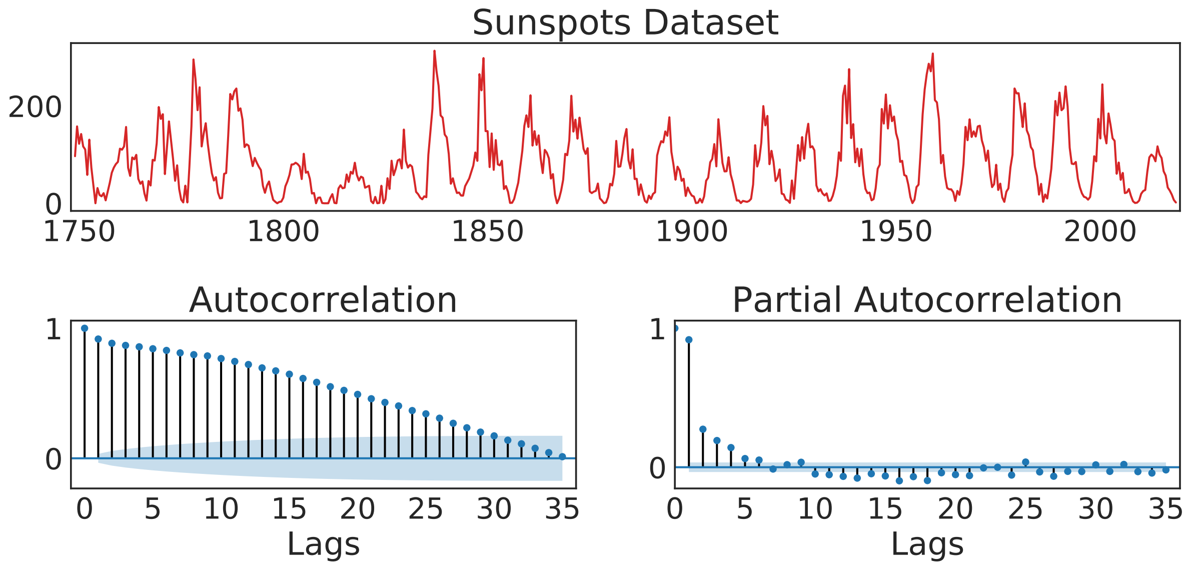

Figure 2.4: Sunspots time series (top) alongside autocorrelation and partial autocorrelation functions.

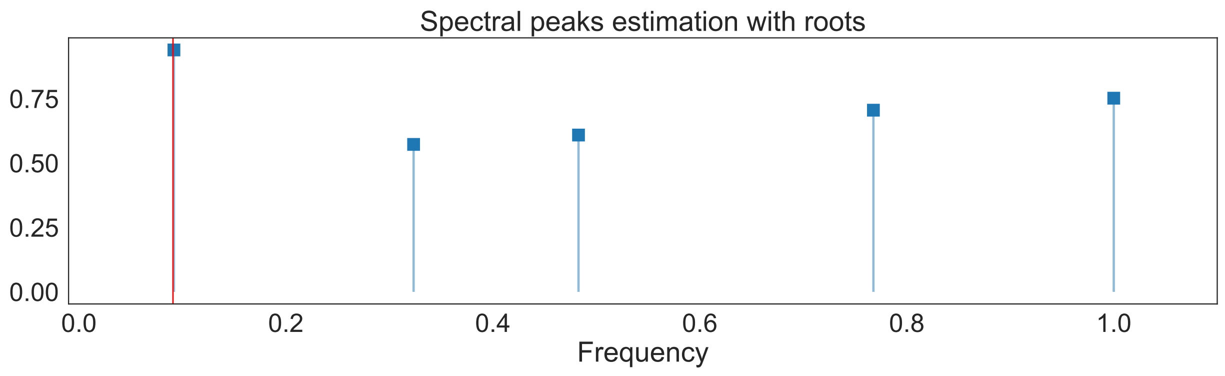

Figure 2.5: Peaks of the PSD computed by BASE for the sunspots time series. The red stem denotes the main peak.

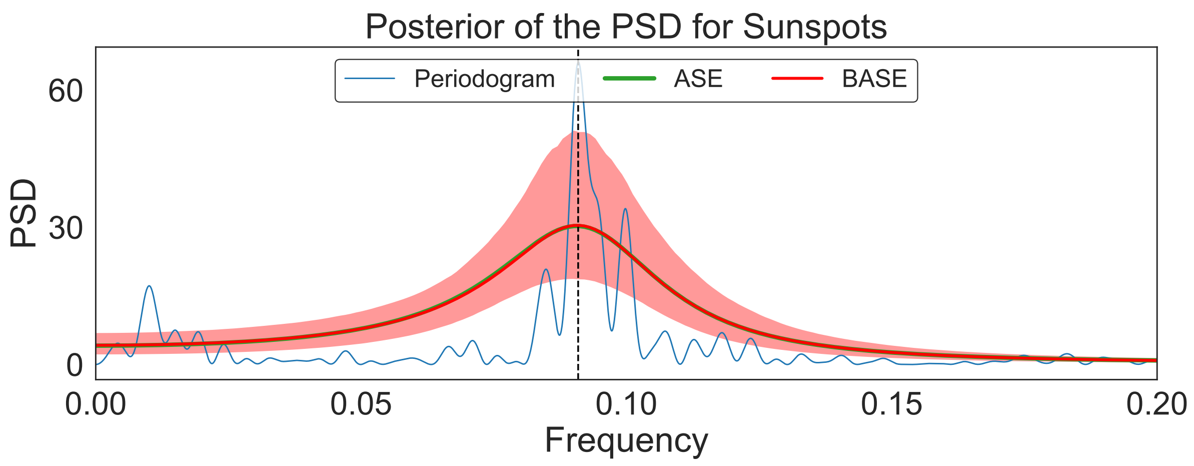

Figure 2.6: PSD estimates for the sunspots series: BASE (95% error bars), ASE and Periodogram. The frequency of the well-known 11-year period is illustrated by a vertical dashed black line.

2.4.4. Performance of the closed-form posterior

First, in order to compare performances, the proposed CF method was implemented on the AR(4) synthethic data generated in *experiment A*, where the real parameters are known. Fig. 2.7 shows the PSD estimate (95% error bars) along with the ASE estimation, the true PSD and the Periodogram. Comparing this results with those shown in Fig. 2.2, we can see the estimation of the mean is slightly better, and the variance of the estimation is slightly reduced.

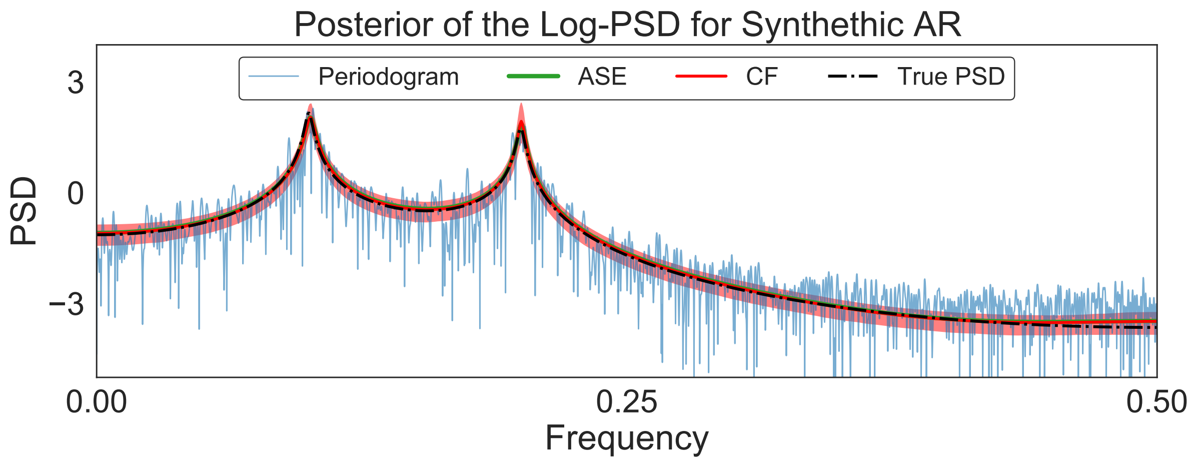

Figure 2.7: Log-PSD estimate for a synthetic AR(4) signal using the CF method; True PSD (dashed black line), Periodogram (blue line), ASE (green line), and proposed CF (red line, 95% error bars).

Next, we used the same synthetic Gaussian process signal generated in *experiment B*, so we can compare the performances of the MCMC method againts the closed-form posterior method on more realistic dataset. Fig. 2.8 shows the PSD estimates (95% error bars for CF) as well as the true PSD of the signal, given by the Fourier Transform of the Laplace kernel. The closed-form method is very similar in performance to its MCMC counterpart, as it also concentrates around the true PSD and its mean captures the estimation of the ASE. Notice that the error bars are slightly better than those given by the MCMC method. This suggest that the posterior distribution we found is actually correct.

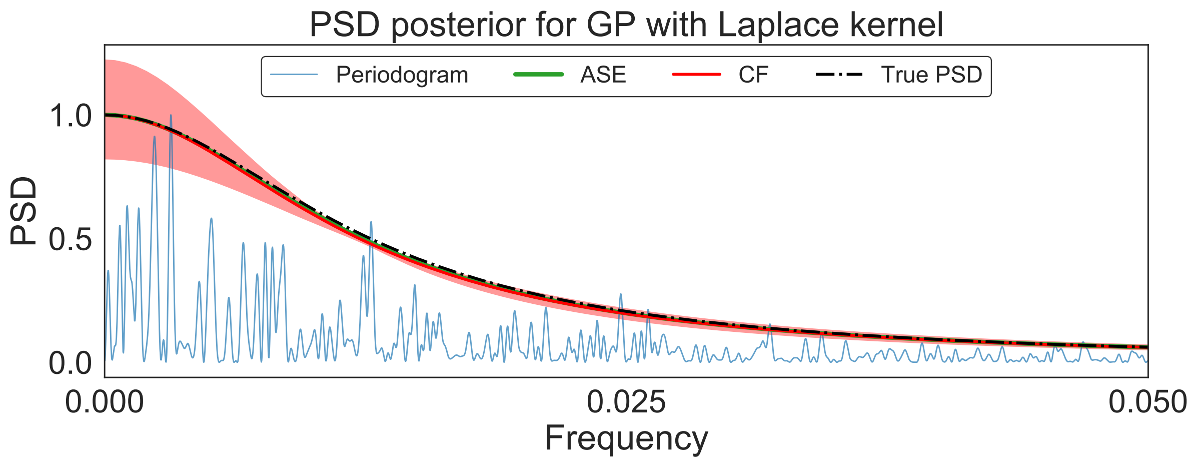

Figure 2.8: Spectral estimation of a GP trajectory: CF shown in red, true PSD in dashed black, Periodogram in blue and ASE in green. The figure only shows the region [0, 0.05], since this is where almost all the spectral content of the data is contained.

In order to validate the closed-form method on real data, we tested it on the sunspots data-set [19], same as we did in *experiment C*. Fig. 2.9 shows the results with the closed-form approach, which again have slightly less variance than its MCMC counterpart. Although the peak frecuency has shifted to the left, the estimation as a whole is still precise and similar to the one given by the MCMC approach. This suggest that the closed-form approach also posses all the good properties the MCMC approach has.

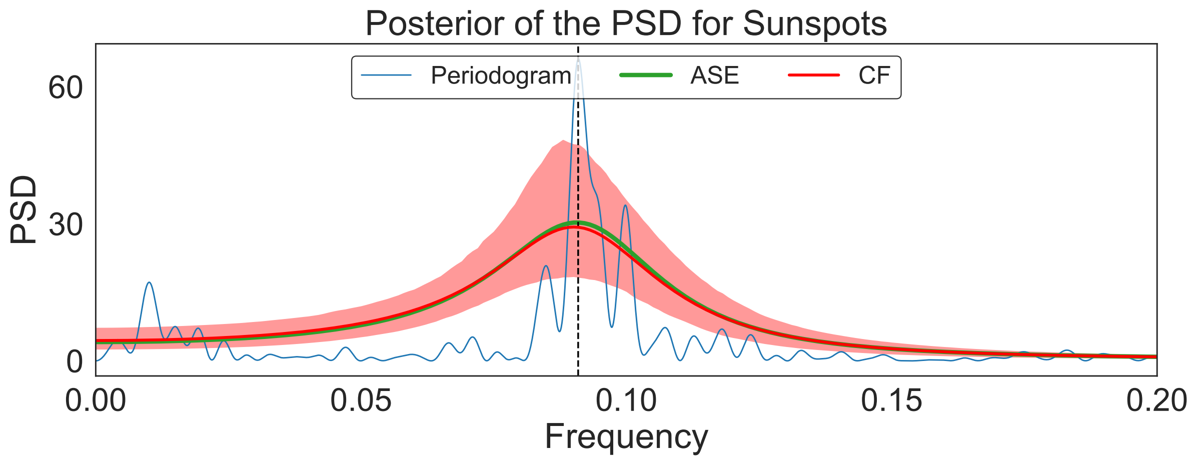

Figure 2.9: PSD estimates for the sunspots series: CF (95% error bars), ASE and Periodogram. The frequency of the well-known 11-year period is illustrated by a vertical dashed black line.

2.5. Conclusion

We have proposed a novel framework termed BASE, a Bayesian approach to autoregressive spectral estimation. BASE exploits the closed-form properties of the AR model to compute power spectral densities (PSD), and also complements it with a probabilistic treatment, thus allowing us to quantify uncertainty in the estimates of the PSDs through the use of confidence intervals.

The description of the proposed method follows the idea that we can sample from the posterior distribution over PSDs (given a set of observations of the time series) by (i) sampling the posterior parameters of an AR model, to then (ii) compute the corresponding PSD of such sample parameter. Using MCMC, we have implemented the proposed BASE method on data generated by an AR model, a Gaussian process model and a real-world astronomical time-series. Through these simulations, we have validated BASE in terms of robustness to model misspecification, unbiasedness, accuracy, and periodicity detection.

On the other hand, although the closed-form approach gives better results than the MCMC approach, its main contribution is the reduction on the computation time. Since it only requires to sample from a given posterior distribution, it takes considerably less time, specially on large datasets. The main setback is that it requires the assumption of certain priors to work, in order to yield a known posterior distribution.

2.6. Future work

Future work includes considering a hierarchical prior to simultaneously identify the model order (e.g., a Dirichlet prior) and extensions to ARMA models to cater for moving-average spectral components and spectra with zeros.

Also, for the closed-form approach, considering other heuristics to determine the hyper-parameters $\mu_0, \lambda, \alpha, \beta$, that include information about the nature of said parameters, may yield better estimations than simply using the grid search.

Chapter 3

Neural networks trained with Cross-entropy loss follow the NTK

3.1. Introduction

Neural Networks are models used in the area of machine learning, which have gained a lot of popularity in recent years due to the impressive results in the multiple task they are used in, such as image recognition, language processing, etc.

However, the number of applications of neural networks have grown much faster than our capacity to understand them, and to this day, most of the inner works of neural networks remains a mystery to be solved, especially in the area of deep learning, where the parameters of the networks lack a proper interpretation.

In an attempt to improve the understanding of deep neural networks, recent works have focused on the importance of the initialization of the parameters of the neural networks. In line with this objective, the study of **Bayesian deep neural networks (BDNN)** have been essential. BDNN are neural networks whose parameters are initialized randomly, according to a prior distribution.

Later, thanks to an specific application of Bayesian deep networks, a new model is born: Infinitely Wide Neural Network. These models are mostly theoretical of course, but it has been proven that a large class of Bayesian deep networks become **Gaussian Processes (GP)** when the width of their layers tends to infinity. This correspondence was first proven for a one-layer feed-forward fully-connected network in [20], and was later extended for multi-layer settings in [21, 22]. Since then, the correspondence between Bayesian deep networks and Gaussian processes has been greatly expanded, proving it for a variety of non-linearities [22, 23] and a variety of architectures, including convolution layers [24, 23, 25, 26], pooling layers [23, 26], skip connections[25], residual networks [24]. In general, any of the usually used architectures, including recurrent networks, or a mixture of said architectures, can be represented by a Gaussian process in the infinite-width limit, as was shown in [27].

In addition to the correspondence of the networks as Gaussian processes, it has been proven that Bayesian deep networks, when initialized randomly and trained using gradient descent, follow a certain behavior described by the **Neural Tangent Kernel (NTK)**[1, 28], which is related to the Kernel of the Gaussian representation of the network.

Although there has been great advances in understanding and exploring these infinite-width limit architectures and the correspondence with their respective Gaussian process, the vast majority of these efforts have centered in changing the architecture of the network, either changing the type of layers or the non-linearities used in the network. There has been little research on the impact that changing the cost function or the training scheme used to train the networks has on this correspondence between BDNN and GP.

Most of the previous works mentioned before only use the mean squared error (MSE) function as their cost function, and use a training scheme based on gradient descent, or stochastic gradient descent. Based on the necessary conditions for the correspondence between BDNN and GP to work, it is natural to think this correspondence will hold for other cost functions and training schemes.

With the idea of proving this correspondence property, the objective of this chapter is to present the contributions of this thesis to the theoretical understanding of fully-connected feed-forward neural networks behavior when the width of the layers tends to infinity, when the cross entropy function is used as the cost function and when the network is trained using stochastic gradient descent. First, I present the necessary definitions and theorems, developed in [1]. Then, I study the properties of the cross-entropy loss, and prove that it fits all the requirements presented in the Background section. Finally, I present a theoretical infinite-width dynamic for the representative function of the network.

The work presented on this chapter, to the best of the author knowledge, is a novel contribution in the field, developed by the author of this thesis. The work done here is based on the theoretic frame developed in [1], but it explores the importance of the cost function, rather than the importance of the network architecture, which is the most common direction that subsequent works explore.

The main contributions of this chapter are:

1. Proving theoretically that the cross-entropy loss fits all the hypothesis needed to study deep neural networks that use such loss function under the NTK scope.
2. Elucidate an equation that describes the evolution of an infinite-width deep neural network during the training process.
3. Ascertain experimentally that deep neural networks, when trained with gradient descent and using the cross-entropy loss, behave, asymptotically in the width, as described by the previously mentioned equation.

3.2. Background

In this section, we will provide a quick review of the most important elements presented in NTK paper[1], in order to have all the ingredients that we will need for our later statements.

3.2.1. Neural Networks

In this work, we will be using the same context used in the NTK paper[1], more specifically, we consider fully-connected neural networks with layers numbered from 0 (input) to L (output), each containing $n_0, \dots, n_L$ neurons, and with a Lipschitz, twice differentiable non-linearity function $\sigma : \mathbb{R} \to \mathbb{R}$, with bounded second derivative.

We will call $\mathcal{F} = \{f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}\}$ the function space, and for a neural network with a fixed set of parameters θ, the function f_θ will be the one representing the network. On this space, we consider the seminorm $|| \cdot ||_{p_{in}}$, defined by:

$$\langle f, g \rangle_{p_{in}} = \mathbb{E}_{x \sim p_{in}}[f(x)^\top g(x)] = \frac{1}{m} \sum_{i=1}^{m} f(x_i)^\top g(x_i). \tag{3.1}$$

The realization function $F^{(L)} : \mathbb{R}^P \to \mathcal{F}$ maps the parameters θ of the network to the representative function f_θ of the network. The parameters consist of the connection matrices $W^{(l)} \in \mathbb{R}^{n_{l+1} \times n_l}$ and biases $b^{(l)} \in \mathbb{R}^{n_{l+1}}$ for $l = 0, \dots, L-1$. The parameters are initialized as independant and identically distributed Gaussian variables. The hyperparameters of this Gaussian distribution (mean and variance) will be defined in the Experiments section.

We consider p_{in} as the empirical distribution on a finite dataset $\{x_1, \dots, x_m\}$, i.e. $p_{in} = \frac{1}{m} \sum_{i=1}^{m} \delta_{x_i}$.

3.2.2. Kernel Gradient

The training of a neural network consist basically in optimizing f_θ in the function space $\mathcal{F}$ with respect to a cost functional $C : \mathcal{F} \to \mathbb{R}$, which in this work, will be the cross-entropy cost.

We define a multi-dimensional Kernel as a function $K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$ with the property $K(x, x') = K(x', x)^\top$. Using a kernel K, we can construct a bi-linear map on $\mathcal{F}$:

$$\langle f, g \rangle_K := \mathbb{E}_{x, x' \sim p_{in}}[f(x)^\top K(x, x') g(x')] = \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} f(x_i)^\top K(x_i, x_j) g(x_j). \tag{3.2}$$

We will say K is positive definite with respect to $|| \cdot ||_{p_{in}}$ if $||f||_{p_{in}} > 0 \Rightarrow ||f||_K > 0$.

We denote by $\mathcal{F}^*$ the space of linear forms $\mu : \mathcal{F} \to \mathbb{R}$ that have the form $\mu = \langle d, \cdot \rangle_{p_{in}}$ for some $d \in \mathcal{F}$. Using the fact that, for a fixed $x \in \mathbb{R}^{n_0}$, a row of the partial application of the kernel $K_{i,\cdot}(x, \cdot)$ is a function in $\mathcal{F}$, we can define a map $\Phi_K : \mathcal{F}^* \to \mathcal{F}$ mapping a dual element $\mu = \langle d, \cdot \rangle_{p_{in}}$ to the function f_μ defined as:

$$f_\mu(x)_i = \mu(K_{i,\cdot}(x, \cdot)) = \langle d, K_{i,\cdot}(x, \cdot) \rangle_{p_{in}} = \frac{1}{m} \sum_{i=1}^{m} d(x_i)^\top K_{i,\cdot}(x, x_i). \tag{3.3}$$

Since the cost functional C only depends on the values of $f \in \mathcal{F}$ at the data points, the functional derivative of C at a point $f_0 \in \mathcal{F}$ can be viewed as an element of $\mathcal{F}^*$. We will denote this element as $\partial_f^{in} C|_{f_0}$, and its dual element in $\mathcal{F}$ as $d|_{f_0}$, such that $\partial_f^{in} C|_{f_0} = \langle d|_{f_0}, \cdot \rangle_{p_{in}}$.

We will define the kernel gradient $\nabla_K C|_{f_0} \in \mathcal{F}$ as $\Phi_K \left(\partial_f^{in} C|_{f_0} \right)$, and its mapping of

$x \in \mathbb{R}^{n_0}$ is:

$$\nabla_K C|_{f_0}(x) = \frac{1}{m}\sum_{i=1}^{m} K(x, x_i) d|_{f_0}(x_i). \tag{3.4}$$

We will say that a time dependant function f_t follows the kernel gradient descent, with respect to kernel K, if it satisfies the relation:

$$\partial_t f_t = -\nabla_K C|_{f_t}. \tag{3.5}$$

Therefore, during the kernel gradient descent, the cost $C(f_t)$ evolves as:

$$\partial_t C|_{f_t} = -\langle d|_{f_t}, \nabla_K C|_{f_t}\rangle_{p_{in}} = -||d|_{f_t}||_K^2. \tag{3.6}$$

From this equation, we conclude that the convergence of C to a critical point is guaranteed if the kernel K is positive definite with respect to p_{in}, since the cost will be strictly decreasing, except at points where $||d|_{f_t}||_{p_{in}} = 0$. Moreover, if C is convex and lower-bounded, then f_t converges to a global minima.

3.2.3. Neural tangent kernel

Following the idea presented in [1, Section 3.1], the authors conclude that performing gradient descent on the cost $C \circ F^{(L)}$ in the parameter space is equivalent to performing kernel gradient descent in the function space $\mathcal{F}$.

This implies that, for neural networks, during training using gradient descent, the network function f_t evolves along the negative kernel gradient:

$$\partial_t f_t = -\nabla_{\Theta_t^L} C|_{f_t}, \tag{3.7}$$

where $\Theta_t^{(L)}$ is the neural tangent kernel (NTK):

$$\Theta_t^{(L)} = \sum_{p=1}^{P} \partial_{\theta_{t,p}} F^{(L)}(\theta_t) \otimes \partial_{\theta_{t,p}} F^{(L)}(\theta_t). \tag{3.8}$$

However, $F^{(L)}$ is not linear, and as a consequence the NTK depends on the parameters θ_t. Since the parameters are changing over time, so does the kernel, which makes the analysis of f_t more difficult.

Proposition 3.1, Theorem 3.1 and Theorem 3.2 help us to solve this problem, by showing that in the infinite-width limit, the NTK is deterministic at initialization and stays constant during training.

Proposition 3.1 *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as $n_1, ..., n_{L1} \to \infty$, the output functions $f_\theta^{(k)}$, for $k = 1, \ldots, L$, tend in* *law to iid centered Gaussian processes of covariance $\Sigma^{(k)}$ defined by recursivity:*

$$\Sigma^{(1)}(x, x') = \frac{1}{n_0} x^\top x' + \beta^2 \tag{3.9}$$

$$\Sigma^{(k+1)}(x, x') = \mathbb{E}_{f \sim GP(0,\Sigma^{(k)})}[\sigma(f(x))^\top \sigma(f(x'))] + \beta^2. \tag{3.10}$$

Theorem 3.1 *For a network of depth L at initialization, with a Lipschitz non-linearity σ, and in the limit as $n_1, ..., n_{L1} \to \infty$, the NTK $\Theta_0^{(L)}$ converges in probability to a deterministic kernel:*

$$\Theta_0^{(L)} \to \Theta^{(L)} \otimes Id_{n_L}, \tag{3.11}$$

where the scalar kernel $\Theta^{(L)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined by recursion:

$$\Theta^{(1)}(x, x') = \Sigma^{(1)}(x, x') \tag{3.12}$$

$$\Theta^{(k+1)}(x, x') = \Theta^{(k)}(x, x')\Sigma'^{(k+1)}(x, x') + \Sigma^{(k+1)}(x, x'), \tag{3.13}$$

where

$$\Sigma'^{(k+1)}(x, x') = \mathbb{E}_{f \sim GP(0,\Sigma^{(k)})}[\sigma'(f(x))^\top \sigma'(f(x'))], \tag{3.14}$$

and σ' denotes the derivative of the non-linearity σ.

When training the network with gradient descent, the parameters are updated according to:

$$\partial_t \theta_{t,p} = \left\langle \partial_{\theta_{t,p}} F^{(L)}(\theta_t), -d|_{f_t} \right\rangle_{p_{in}}. \tag{3.15}$$

Assuming that the integral $\int_0^T ||d|_{f_t}||_{p_{in}} dt$ stays stochastically bounded when the widths tend to infinity, we have the following result:

Theorem 3.2 *Assume σ is a Lipschitzt, twice differentiable non-linear function, with bounded second derivative. For any $T \in \mathbb{R}_+$ such that $\int_0^T ||d|_{f_t}||_{p_{in}} dt$ stays stochastically bounded, as $n_1, ..., n_{L1} \to \infty$, we have uniformly for $t \in [0, T]$:*

$$\Theta_t^{(L)} \to \Theta^{(L)} \otimes Id_{n_L}, \tag{3.16}$$

and as a consequence, in this limit, the dynamics of f_t is described by:

$$\partial_t f_t = \Phi_{\Theta^{(L)} \otimes Id_{n_L}} \left(\langle -d|_{f_t}, \cdot \rangle_{p_{in}} \right). \tag{3.17}$$

3.3. Cross entropy loss function

In this section, we will focus in a specific case, a neural network used for classification problems, trained with gradient descent and using a cross-entropy loss. The objective is to verify that this specific setup fulfills all the hypothesis mentioned in the Background section, and so, can be studied and understood through the NTK approach.

Since we now consider a classification problem, any input $x \in \mathbb{R}^{n_0}$ of the neural network will be associated with one of n_L different classes. The true class of an input x will be one-hot encoded through the function $f^* : \mathbb{R}^{n_0} \to \{0, 1\}^{n_L}$, ie, $f^*(x)$ will be a vector of 0's, and one 1 value in the position representing the true class of x.

As was mentioned in the Background section (3.2), the cost function C we will consider is the cross-entropy:

$$C(f) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n_L}(f^*(x_i))_j \log(f(x_i)_j), \tag{3.18}$$

where $\{(x_i, f^*(x_i))\}_{i=1}^{m}$ are the data points, used to train the network.

In this case, the network function $f_t : \mathbb{R}^{n_0} \to (0,1)^{n_L}$ will be such that $f(x)_j$ represents the probability of x belonging to class j, for $j = 1, \dots, n_L$.

3.3.1. C is convex and non negative

First, we express the function C in a form where it is easier to understand why it is convex and non-negative:

$$\begin{aligned} C(f) &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n_L}(f^*(x_i))_j \log(f(x_i)_j) \\ &= \frac{1}{m}\sum_{i=1}^{m} f^*(x_i)^\top(-\log(f(x_i))) \\ &= \langle f^*, -\log(f(\cdot))\rangle_{p_{in}} \end{aligned} \tag{3.19}$$

Here $\log(f(x))$ means the logarithm is applied at each coordinate of the vector $f(x)$.

Thus, C is defined by the inner product p_{in}, so it is linear in $-\log(f)$ which is a convex function on f. Therefore, by composition, C is convex in f.

To show that it is non-negative, one has to remember that $f : \mathbb{R}^{n_0} \to (0,1)^{n_L}$ and $f^* : \mathbb{R}^{n_0} \to \{0,1\}^{n_L}$, so $-\log(f)$ is a positive function, and the inner product between two non-negatives functions is non-negative.

3.3.2. Direction of training d_t

Suppose that we are going to train the neural network for a certain amount of epochs, then the output function f becomes time dependant, as it is expected that it changes over time due to the training procedure. Therefore, we are calling it f_t to make this dependence explicit.

Our main goal is to describe the function f_t at all times, to understand how the neural network changes over time. To achieve this, we need to discover the training direction d_t, to understand the dynamic of f_t.

To do so, we first have to look at how the cost function changes over time, since the output function f_t is changing:

$$\begin{aligned}
\partial_t C|_{f_t} &= \partial_t \left(-\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_L} (f^*(x_i))_j \log(f_t(x_i)_j) \right) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_L} (f^*(x_i))_j \partial_t (\log(f_t(x_i)_j)) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_L} (f^*(x_i))_j \frac{1}{f_t(x_i)_j} \partial_t (f_t(x_i)_j) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(\frac{f^*}{f_t}(x_i) \right)^\top \partial_t f_t(x_i) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(\frac{f^*}{f_t}(x_i) \right)^\top \cdot (-\nabla_K C|_{f_t}(x_i)) \\
&= -\left\langle -\frac{f^*}{f_t}, \nabla_K C|_{f_t} \right\rangle_{p_{in}}
\end{aligned} \tag{3.20}$$

and taking into account the theory presented in the previous section, we conclude that $d|_{f_t} = -\frac{f^*}{f_t}$. Moreover, in gradient descent, the training direction is $d_t = -d|_{f_t} = \frac{f^*}{f_t}$. Notice that the direction found makes sense with the goal of the network, since it makes f_t to grow only in the direction of the true class of the inputs.

3.3.3. $\int_0^T ||d_t||_{p_{in}} dt$ stays stochastically bounded.

One last requirement is that the training direction is well defined over all the training procedure. Notice that $||d_t||_{p_{in}} = ||\frac{f^*}{f_t}||_{p_{in}}$ is strictly decreasing over training, due to the fact that f_t becomes more similar to f^* thanks to the training direction. Therefore, the integral is bounded by $T||d_0||_{p_{in}}$.

As a consequence of all the previous results exposed, and following Theorem (3.2), we conclude that in the infinite-width limit of the neural network, the dynamics of f_t is described by:

$$\partial_t f_t = \Phi_{\Theta^{(L)} \otimes Id_{n_L}} \left(\left\langle \frac{f^*}{f_t}, \cdot \right\rangle_{p_{in}} \right). \tag{3.21}$$

Expanding the terms in the equation, we obtain a new expression that is easier to analyze:

$$\partial_t f_t(x) = \sum_{j=1}^{m} \Theta^{(L)} \otimes Id_{n_L}(x, x_j) \cdot \frac{f^*}{f_t}(x_j). \tag{3.22}$$

And looking at it coordinate-wise:

$$\partial_t f_t(x)_i = \sum_{j=1}^{m} \sum_{s=1}^{n_L} (\Theta^{(L)} \otimes Id_{n_L})_{i,s}(x, x_j) \frac{f^*(x_j)_s}{f_t(x_j)_s}, \quad \forall i \in \{1, \dots, n_L\}. \tag{3.23}$$

Since the NTK $\Theta^{(L)} \otimes Id_{n_L}$ is a diagonal matrix, there is only one term in the inner summation.

$$\partial_t f_t(x)_i = \sum_{j=1}^{m} (\Theta^{(L)} \otimes Id_{n_L})_{i,i}(x, x_j) \frac{f^*(x_j)_i}{f_t(x_j)_i}, \quad \forall i \in \{1, \dots, n_L\}. \tag{3.24}$$

Recall that f^* is the true class function, therefore $f^*(x_j)_i = 1$ if x_j belong to the i class, otherwise it is equal to 0.

Looking at the previous equation, we can see that during training, the network decides the class of a given input x by looking at the class of the data points and their correlation with x. More specifically, the probability of assigning x to a class i, will only depend on the data points that belong to the class i, and their correlation to x.

3.4. Experiments

The following numerical experiments will compare fully-connected neural networks of **various widths** to their theoretical infinite-width limit.

For a fixed-width neural network, we will look at the empirical NTK's of the network at two different time stamps: $t_0 = 0$, ie, at initialization; and $t_f = 1000$, ie, at the end of the training process. The respective empirical NTK's at these timestamps will be $\Theta_0^{(L)}$ and $\Theta_{t_f}^{(L)}$, where L is the depth of the network. These empirical NTK's are calculated following the formula in equation (3.8).

Simultaneously, we will calculate the infinite-width limiting NTK $\Theta^{(L)} \otimes Id_{n_L}$ of the network, according to the formulas given by Theorem (3.1).

Following the statements in Theorem 3.1 and Theorem 3.2, the expected behavior is that these empirical NTK's, at both timestamps t_0 and t_f, will tend to the infinite-width limiting NTK as the width of the hidden layers of the network increases. The similarity or agreement between the empirical NTK's and the infinite-width limiting NTK will be measured with the Euclidean distance.

For two matrices A, B with the same dimensions $n \times m$, the Euclidean distance between them is calculated as:

$$D(A, B) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} (A_{i,j} - B_{i,j})^2}. \tag{3.25}$$

Also, sice Theorem (3.2) states that the infinite-width limiting NTK is time invariant, we are also expecting that the empirical NTK's of the network vary less as the width of the hidden layers increases.

We will make 2 experiments, and in both of them we will construct 4 fully-connected neural networks with 2 hidden layers of the same width, so the depth of the networks will be $L = 3$. The width of the hidden layers will be equal to 2^w, with $w \in \{2, 5, 7, 9\}$ in both experiments, and to identify which of the neural networks are we talking about, we will name them by referencing their width hyper-parameter w. So, for example, the neural network NN2 will be the network in which $w = 2$; NN5 will be the network in which $w = 5$, and so on. The non-linear activation function σ will be the error function for both experiments, so $\sigma(x) = Erf(x)$.

Recalling the parameters of the network W, b are initialized as i.i.d. Gaussians, the mean of this distribution will be $\mu = 0$ for both W and b in all the L layers. The standard deviations for the networks parameters W, b are chosen as $\sigma_W = 1$ for the weights, and $\sigma_b = 0.05$ for the biases, in all the layers of the networks.

The first experiment, which we will refer to as SIMPLE, is a rather simple classification problem, that consist on classifying real numbers according to their sign. More specifically, the inputs x will be real numbers in the $[-\pi, \pi]$ interval, and the classes $f^*(x) \in \mathbb{Z}^2$ are $(1, 0)$ if x is negative, and $(0, 1)$ if it is non-negative. Therefore, the input dimension $n_0 = 1$, and the output dimension $n_L = 2$. The data will be divided in 10 training points and 5 test points.

The second experiment, which we will refer to as MNIST, is the famous MNIST classification problem, which consists on classifying a set of 28×28 pixels, white and black images of handwritten digits, therefore, there are 10 different classes for the input images. Thus, the input dimension is $n_0 = 28 \cdot 28$ and the output dimension is $n_L = 10$. We will take 50 training points and 50 test points.

The neural networks will be trained with stochastic gradient descent, over 1000 steps with a learning rate equal to 1.

3.4.1. Convergence of the empirical NTK of the neural network to a fixed limiting kernel

First, we will check the behavior of the Neural Tangent Kernel, as described in equations (3.11) and (3.16).

Figure (3.1) and Figure (3.2) shows the values of the Neural Tangent Kernel for the SIMPLE experiment and the MNIST experiment respectively, on the training data points, at initialization and at the end of the stochastic gradient descent training process. As we can see, as the hidden layers width grows, the empirical NTK at initialization converges to the theoretical infinite-width limiting NTK, as expected from equation (3.11). Moreover, over the course of the training process, the empirical NTK changes over time and drifts away from the infinite-width limiting NTK. This effect is most notorious for the two most narrow networks in both experiments, and is also noticeable that the empirical NTK of the two wider networks remains almost constant, and very similar to the theoretical infinite-width limiting NTK.

The dependence in time of the empirical NTK was stated in the Background section (3.8), so the behavior of the empirical NTK in the experiments was expected. What is more surprising is the fact that this dependence over time is gradually lost when the width of the hidden layers tends to infinity. Theorem (3.2) only states that the infinite-width limiting NTK is time independent, but does not refer to finite-width NTK's. The experiments show

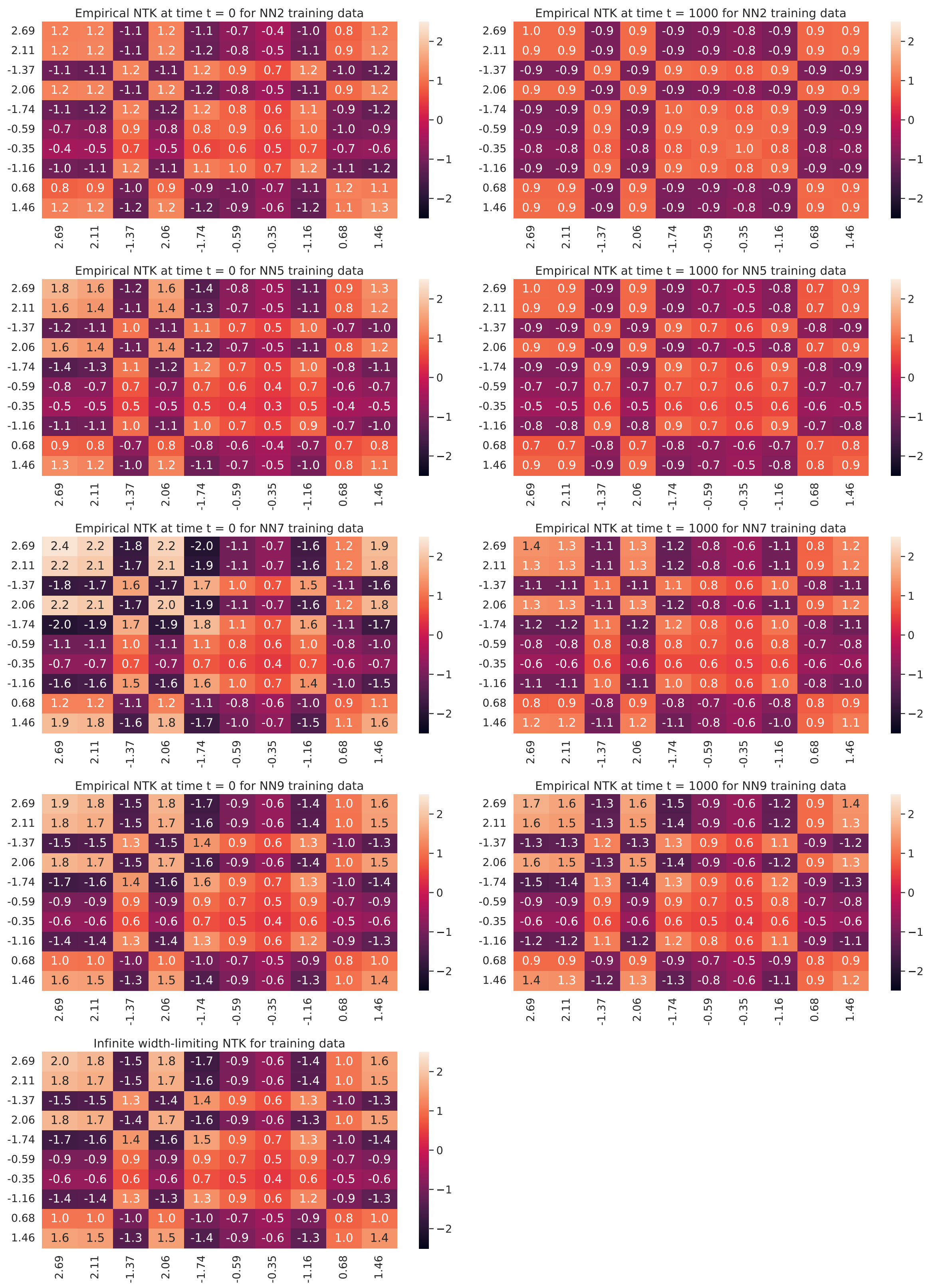

Figure 3.1: Values of the empirical NTK for different layer widths for the SIMPLE experiment. Left: Empirical NTK of the network at training time t=0. Right: Empirical NTK at training time t=1000. Last: Infinite-width limiting NTK, which is independent from time.

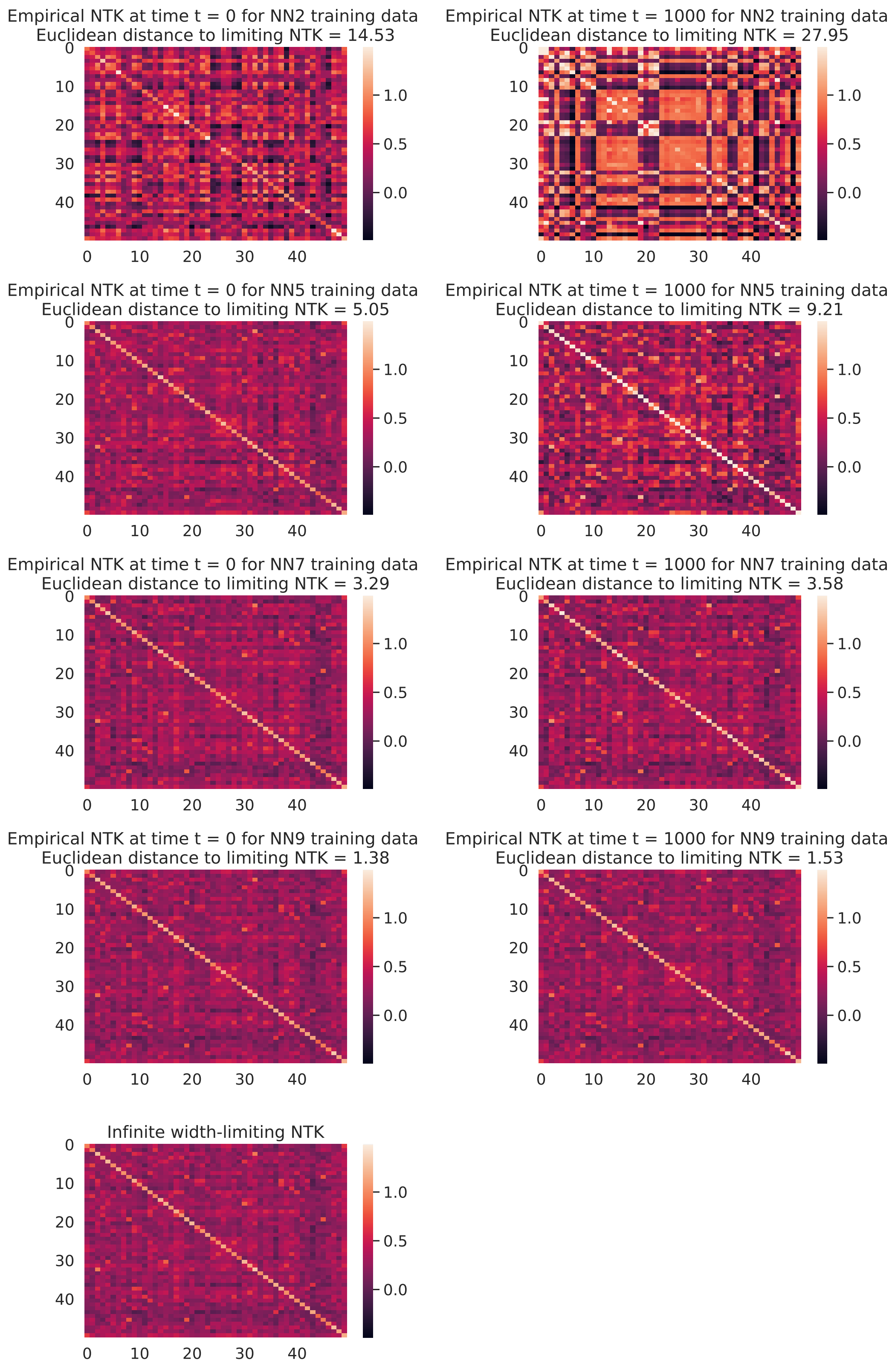

Figure 3.2: Values of the empirical NTK for different layer widths for the MNIST experiment. Left: Empirical NTK of the network at training time t=0. Right: Empirical NTK at training time t=1000. Last: Infinite-width limiting NTK, which is independent from time.

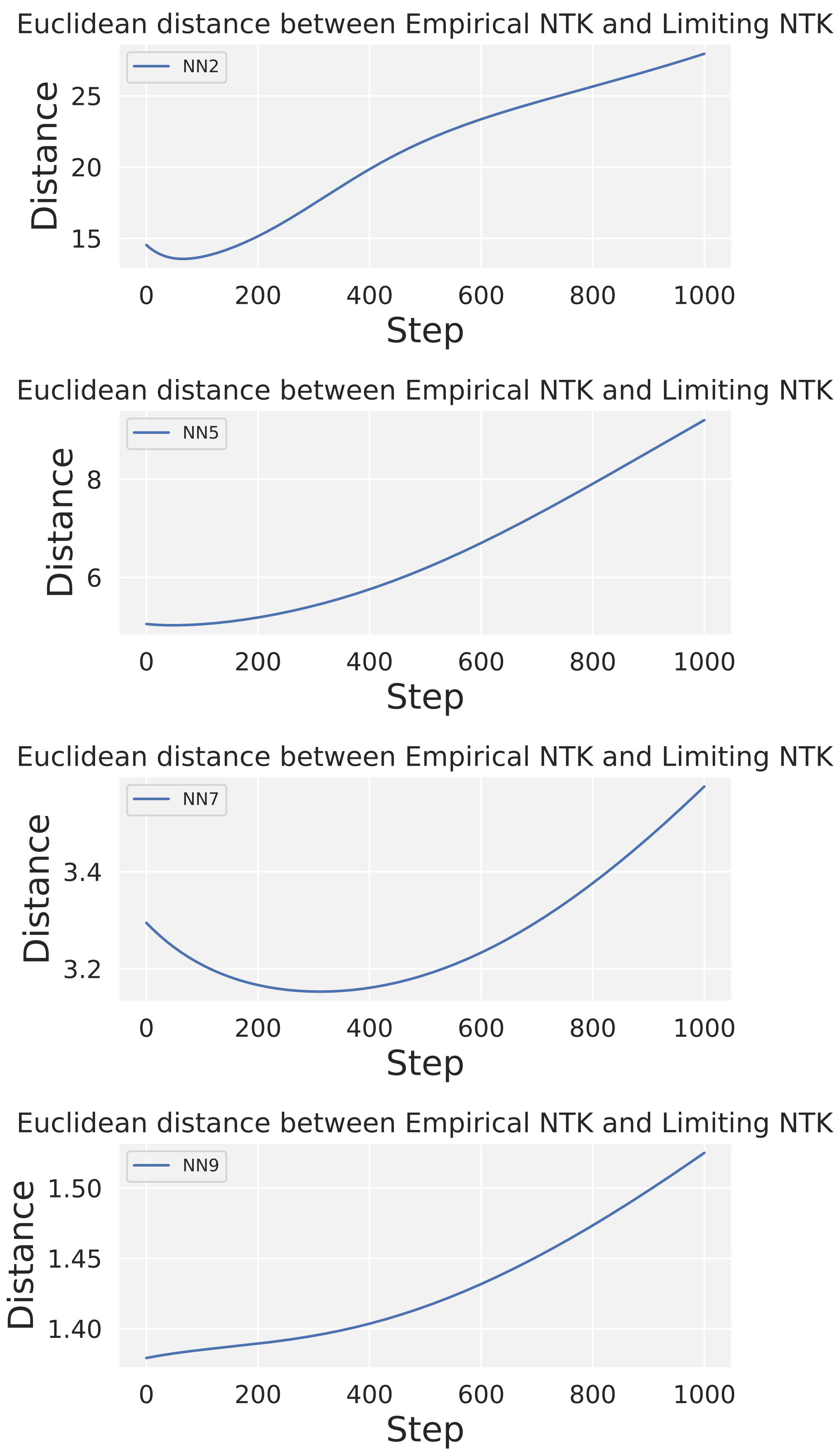

Figure 3.3: Values of the Euclidean distance between the empirical NTK on a fixed training step and the infinite-width limiting NTK for the MNIST experiment.

that the empirical NTK of wider networks varies less in time, which means that the evolution of the training direction for wider networks is less time dependant.

In Figure (3.3), for the MNIST experiment, the Euclidean distance between the empirical NTK and the infinite-width limiting NTK is calculated for each training step. Notice that the scale on each graph is different, which is important for the interpretation of the results. As we can see, as a general result, the distance between the empirical NTK and the infinite-width limiting NTK is smaller for wider neural networks, which further supports the convergence stated in equation (3.16).

Even more, if we fixate in a single time stamp $t^* \in [t_0, t_f]$, for example we could take $t^* = 600$, we can see that as the width of the network increases, the Euclidean distance between the empirical NTK and the infinite-width limiting NTK decreases for that singular time stamp. This same effect occurs for every $t^* \in [t_0, t_f]$, which further supports the second statement given in Theorem (3.2), that is, that the convergence of the NTK is occurring uniformly for every $t^* \in [t_0, t_f]$.

A new interesting phenomena that can be seen in the graphs, is the fact that the empirical NTK remains almost constant for the first steps of the training, independent of the width of the network. In general, we can see that for all the studied networks, previous to the step 70 of training, the distance between the empirical NTK and the limiting NTK suffer minor changes, which indicates that the empirical NTK is not changing too much. After that, and specially after the step 100 of training, we can see an important increase on the distance between the empirical NTK and the limiting NTK, due to the fact that the empirical NTK suffer major changes, as was seen in Figure (3.2).

Recalling that the time dependence of the empirical NTK is inherited from the time dependence of the parameters θ_t of the neural network, as was stated in equation (3.8), allows us to conclude that if the empirical NTK is not changing on the first steps of the training, then the parameters of the neural network are not changing either. This result suggest that the learning process does not occur immediately as we start training the network, but takes some warm-up steps to actually start making some changes on the network.

3.4.2. Convergence of the finite neural network to the infinite-width limit

We will now study the empirical behavior of the neural network itself against the theoretical behavior proposed in equation (3.22). We will compare the values of the cross entropy loss for each one of the constructed finite-width neural networks against the values of the cross entropy loss for a theoretical infinite-width neural network. Recall that, since our interest is to examine the impact of the layer width on the behavior of the neural network, the only architectural difference between the different finite neural networks and, of course, their infinite counterpart, is the width of their hidden layers.

Since we cannot construct an infinite-width neural network, we will simulate one using the ODE (3.22) that describes the evolution of the network function f_t. More specifically, we can construct the NTK of the network since we know its architecture, and therefore, we can simulate numerically the ODE to obtain the predictions of the network at a given time t, on the training and test data points, and with these predictions, we can calculate the cross-entropy loss of the network at the given time t.

For the finite-width neural networks, the process is very simple, we construct the neural network, train it using gradient descent, and at each step t of the training procedure, we will output the predictions of the network on the training and the test data points. Using these predictions, we can calculate the cross-entropy loss of the neural network at time t.

To make the process simpler, the calculation of the NTK's and the numerical solving of the network's ODE were done using the Neural Tangents package ([29], [30]), a specialized package design to work with infinite-width neural networks and also their finite counterparts.

Figure (3.4) and Figure (3.5) shows the evolution of the values of the cross-entropy loss during the training process, for the four neural networks constructed, in the SIMPLE experiment and the MNIST experiment respectively. As we can see, in the narrowest network (NN2), the evolution of the cross-entropy loss of the network is very dissimilar to the behavior of the infinite-width limiting network. But as the width of the network increases, the evolution of the networks start to look alike the evolution of their infinite-width counterpart. This is an expected behavior, due to the fact that the empirical NTK is changing over time for the narrow networks, and even more, is moving away from the infinite-width limiting NTK, as we saw in the previous Figure (3.2). For the two widest networks (NN7, NN9), the NTK remains almost constant and similar to the infinite-width limiting NTK, as we stated in the previous Figure (3.2), so the dynamic of the representative function f_t is well described by the equation (3.22), and therefore, the losses of these finite-width networks and their infinite-width counterpart are quite similar.

An interesting observation, is the fact that the finite-width neural networks have worse performance than their infinite-width counterpart, in terms of their cross-entropy loss, for all finite hidden layer width sizes; and always have better performance on the training data than on the test data. This phenomena was observed in other related works ([21]), but using different network architectures and a different loss function.

This arises the possibility of using the infinite-width limiting neural networks as benchmark for performance in the field of deep learning, when working with finite neural networks. Although, one must be careful, since the mentioned phenomena is sensitive to architectural choices, as was seen in ([21]), so is important to remember the configuration that we are using for these experiments: fully-connected feed-forward neural networks, trained using gradient descent and cross-entropy loss.

3.5. Conclusion

We developed an equation (3.22) that describes the evolution dynamic of the network function f when the loss functions is the cross entropy loss, when the network is trained using gradient descent. Moreover, we were able to interpret this equation, and have an insight

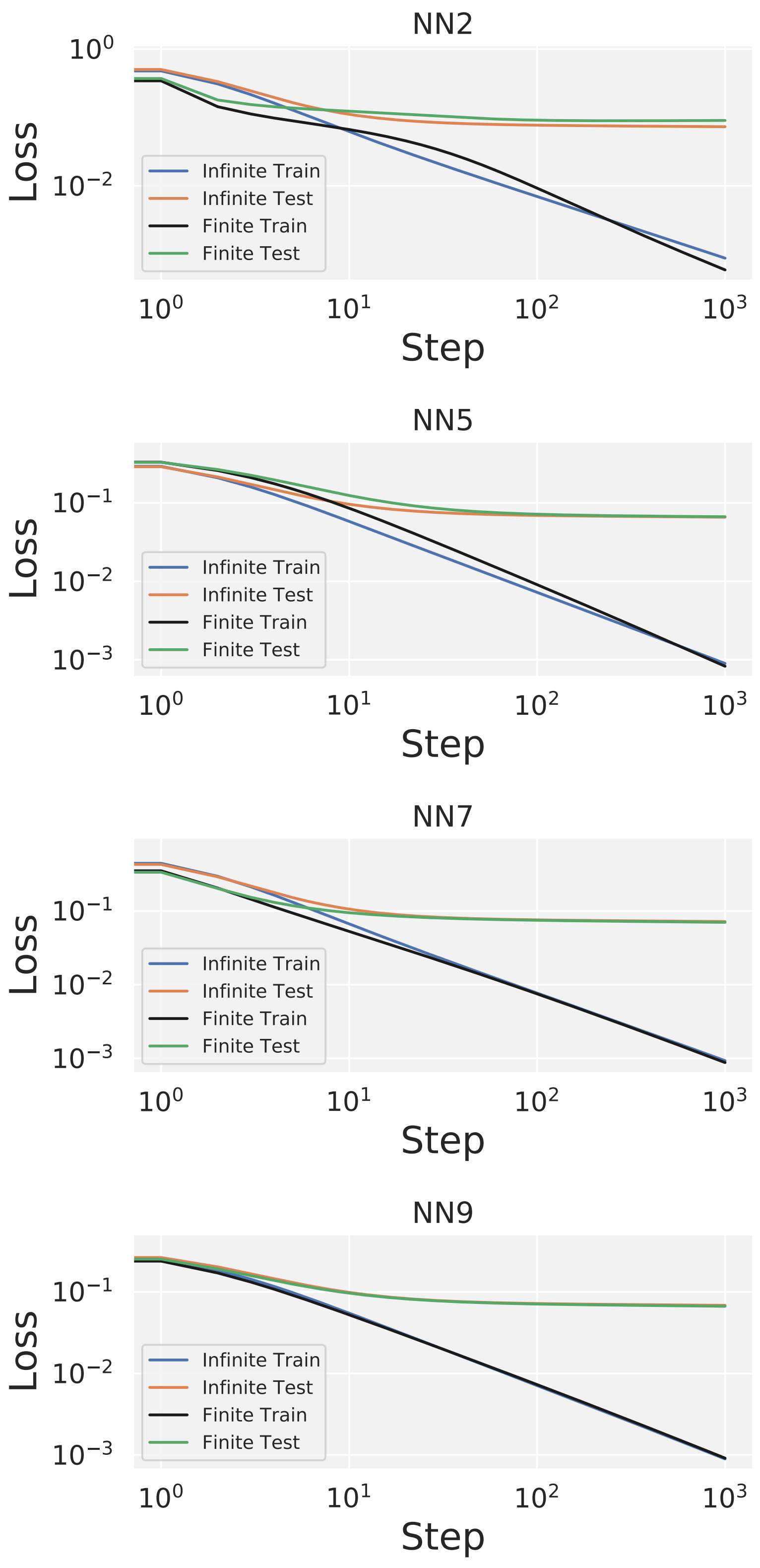

Figure 3.4: Values of the cross-entropy loss of the neural network over the gradient descent training process for the SIMPLE experiment, in logarithmic scale. From top to bottom, the networks are ordered in increasing layer width size. The curves show the losses for the corresponding finite width network, and their infinite-width limiting counterpart, for the training and test data points.

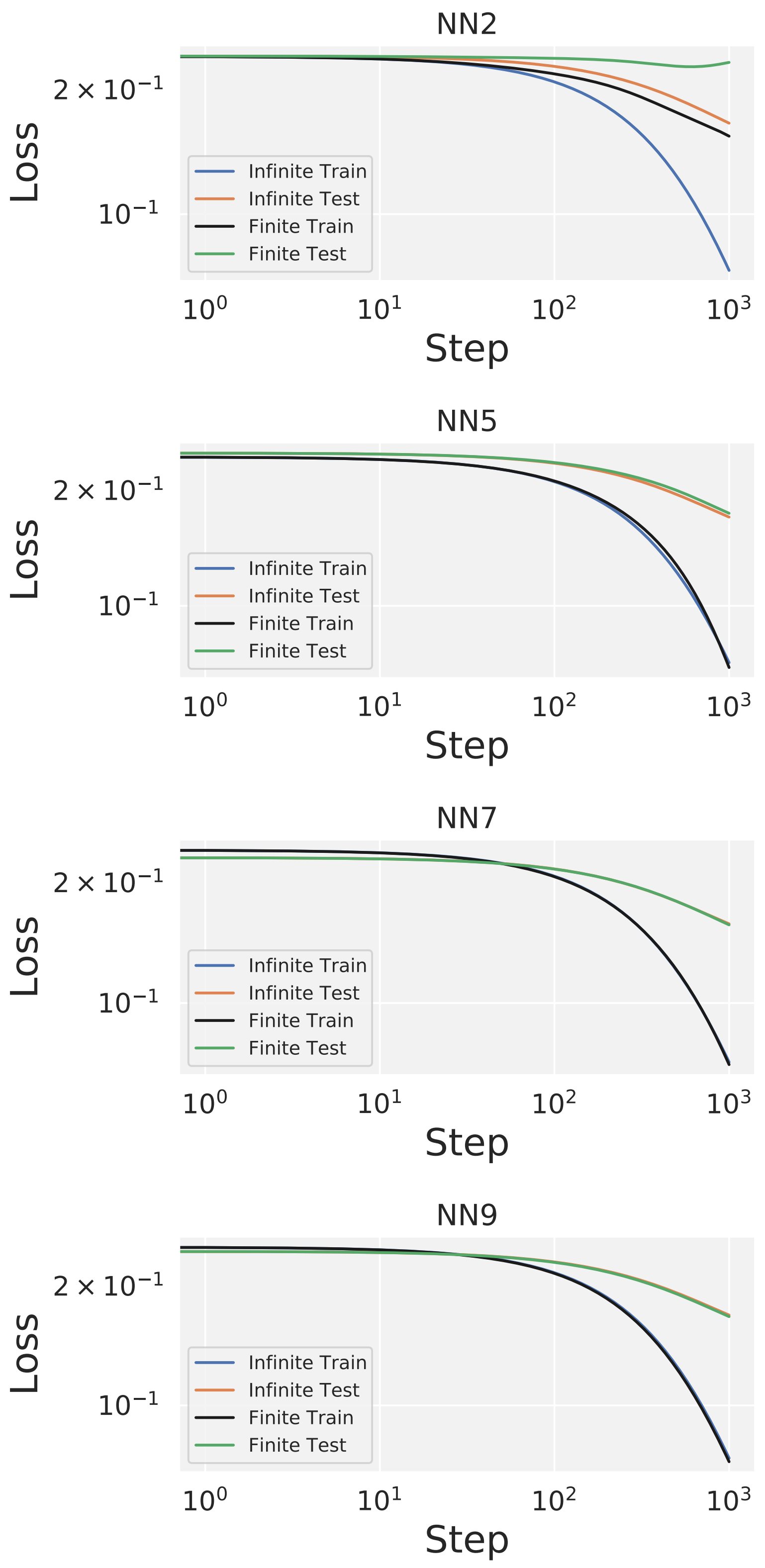

Figure 3.5: Values of the cross-entropy loss of the neural network over the gradient descent training process for the MNIST experiment, in logarithmic scale. From top to bottom, the networks are ordered in increasing layer width size. The curves show the losses for the corresponding finite width network, and their infinite-width limiting counterpart, for the training and test data points.

of how the data, and their correlations are used in the training process.

We also checked that for neural networks with finite hidden layers width, the Neural Tangent Kernel is time dependant, as was stated in [1], and that as the width of the hidden layers of the network grows, it reaches an infinite-width limiting kernel, which is independent from time. This convergence is uniformly for each step t of the training process.

One of the observations made in the last experiment was the fact that the infinite-width limit network always had better training and test score than his finite counterpart, independent of the width of the network layers. This result creates an interesting and more practical application, the infinite-width limit of neural networks can be used as a benchmark for test or training score, at least in the case studied here, which is a feed-forward fully-connected deep neural network. This observation is made in similar works [21] but only for the mean squared error loss, which is different from the case studied here.

3.6. Future work

Following the work made in this thesis, some novel contributions would be:

- To see if other networks, with richer architectures, follow a similar pattern when trained with gradient descent under the cross-entropy loss.
- Experimenting with other loss functions, other than cross-entropy and mean squared error, could be of value, since it could open up the possibility of interpreting other type of problems under the infinite-width limit scope.

Both ideas are a natural follow up to the work done in this thesis, and both of them are reasonable, given the necessary hypothesis for the NTK to converge.

First, lets notice from Theorem (3.1) that the result stated in the theorem does not depend on the cost function, but on the activation function. Although that result only holds for the architecture studied in the paper, which is a feed-forward fully-connected deep neural network, more recent works have proved the convergence at initialization for more richer architectures, including convolution layers, pooling layers, residual connections and recurrent networks.

The more interesting question is whether the convergence of the NTK holds even during the training process or not, as stated in Theorem (3.2). Notice this result now depends also on the direction of training d_t, which in turn depends on the cost function C, so in this case, the decision of which cost function will be used is important.

Since we already proved that the cross-entropy loss function is convex, lower-bounded and defines a training direction d_t that is stochastically bounded, and this properties are independent of the architecture choice, then is reasonable to assume that the equation (3.21) will hold even for other architectures. The only element that should change is the limiting NTK $\Theta^{(L)} \times Id_{n_L}$, since it is architecture dependent.

This last statement is crucial, since depending on the limiting NTK $\Theta^{(L)} \times Id_{n_L}$, the equation (3.21) describing the evolution of f_t could be infeasible.

On the other hand, in order to try a different cost function, we will have to prove the 3 properties that we proved for the cross-entropy:

1. Convexity.
2. Lower-bounded.
3. Defines a training direction d_t that is stochastically bounded in time.

Of the 3 properties, the most important is the third one, since it is needed for the equation that describes the evolution of f_t (3.21) to be well defined for every $t \in [0, T]$.

The other 2 properties (convexity and lower-bound) are actually not strictly necessary. They are required for the neural network to converge to a critical point of minimal cost during training. This is important if we want to use this theory on real world applications, since even if we can define the equation for the evolution of f_t, it will not be useful if during the training process the loss is not descending, ie, the network is not improving. Or even if the network is improving, if the cost function C is not bounded, we could be training the network for an infinite amount of time without ever reaching a final optimal state.

Bibliography

[1] A. Jacot, F. Gabriel, and C. Hongler, “Neural tangent kernel: Convergence and generalization in neural networks,” *Advances in neural information processing systems*, vol. 31, 2018.

[2] C. Park, D. Looney, P. Kidmose, M. Ungstrup, and D. P. Mandic, “Time-frequency analysis of EEG asymmetry using bivariate empirical mode decomposition,” vol. 19, pp. 366–373, 2011.

[3] G. Vazquez-Vilar, R. López-Valcarce, C. Mosquera, and N. González-Prelcic, “Wideband spectral estimation from compressed measurements exploiting spectral a priori information in cognitive radio systems,” in *Proc. of IEEE ICASSP*, pp. 2958–2961, 2010.

[4] R. Davis and P. Brockwell, *Time Series: Theory and Methods.* Springer-Verlag New York, 1991.

[5] N. Lomb, “Least-squares frequency analysis of unequally spaced data,” *Astrophysics and Space Science*, p. 447–462, 1976.

[6] N. Choudhuri, S. Ghosal, and A. Roy, “Bayesian estimation of the spectral density of a time series,” *Journal of the American Statistical Association*, vol. 99, no. 468, pp. 1050–1059, 2004.

[7] G. Parra and F. Tobar, “Spectral mixture kernels for multioutput Gaussian processes,” in *Proc. of NIPS*, pp. 6681–6690, 2017.

[8] D. Williams and V. Madisetti, eds., *Digital Signal Processing Handbook.* Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 1997.

[9] F. Tobar, T. Bui, and R. Turner, “Learning stationary time series using Gaussian processes with nonparametric kernels,” in *Proc of NIPS*, pp. 3483–3491, 2015.

[10] F. Tobar, “Bayesian nonparametric spectral estimation,” in *Proc. of NIPS*, 2018. in press.

[11] Y. Wang, R. Khardon, and P. Protopapas, “Nonparametric Bayesian estimation of periodic light curves,” *The Astrophysical Journal*, vol. 756, p. 67, aug 2012.

[12] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan, “An introduction to MCMC for machine learning,” *Machine Learning*, vol. 50, no. 1, pp. 5–43, 2003.

[13] A. Cuevas, S. López, D. Mandic, and F. Tobar, “Bayesian autoregressive spectral estimation,” in *2021 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pp. 1–7, 2021.

[14] G. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control.* 1990.

[15] P. Halmos, *Measure Theory.* Graduate Texts in Mathematics, Springer New York, 1976.

[16] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[17] M. Homan and A. Gelman, "The No-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo," vol. 15, 2014.

[18] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3," *PeerJ Computer Science*, vol. 2, p. e55, 2016.

[19] SILSO World Data Center, "The International Sunspot Number," *International Sunspot Number Monthly Bulletin and online catalogue*, 1749-2017.

[20] R. M. Neal, "Priors for infinite networks," in *Bayesian Learning for Neural Networks*, pp. 29–53, Springer, 1996.

[21] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv preprint arXiv:1711.00165*, 2017.

[22] A. G. d. G. Matthews, M. Rowland, J. Hron, R. E. Turner, and Z. Ghahramani, "Gaussian process behaviour in wide deep neural networks," *arXiv preprint arXiv:1804.11271*, 2018.

[23] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Bayesian deep convolutional networks with many channels are gaussian processes," *arXiv preprint arXiv:1810.05148*, 2018.

[24] A. Garriga-Alonso, C. E. Rasmussen, and L. Aitchison, "Deep convolutional networks as shallow gaussian processes," *arXiv preprint arXiv:1808.05587*, 2018.

[25] A. Andreassen and E. Dyer, "Asymptotics of wide convolutional neural networks," *arXiv preprint arXiv:2008.08675*, 2020.

[26] A. Garriga-Alonso and M. van der Wilk, "Correlated weights in infinite limits of deep convolutional neural networks," in *Uncertainty in Artificial Intelligence*, pp. 1998–2007, PMLR, 2021.

[27] G. Yang, "Wide feedforward or recurrent neural networks of any architecture are gaussian processes," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[28] 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*, vol. 32, 2019.

[29] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz, "Neural tangents: Fast and easy infinite neural networks in python," in *International Conference on Learning Representations*, 2020.

[30] R. Novak, J. Sohl-Dickstein, and S. S. Schoenholz, "Fast finite width neural tangent kernel," in *International Conference on Machine Learning*, 2022.