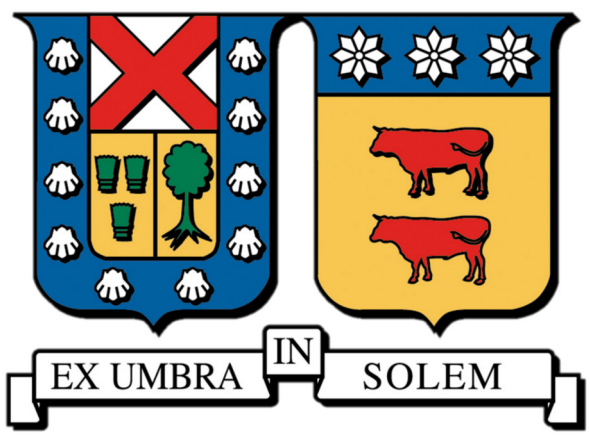
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



PEAK DETECTION OF SPECTRALLY-OVERLAPPED FIBER BRAGG GRATINGS USING AN UNSUPERVISED CONVOLUTIONAL NEURAL NETWORK AUTOENCODER

Tesis de grado presentada por
Gabriel Ignacio Rudloff Barison,
como requisito parcial para optar al titulo de
Ingeniero Civil Electrónico
y para optar al grado de
Magíster en Ciencias de la Ingeniería Electrónica
Mención Telecomunicaciones y Procesamiento de Señales.

MARZO 2023

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO, CHILE

TÍTULO DE LA TESIS:
PEAK DETECTION OF SPECTRALLY-OVERLAPPED FIBER BRAGG GRATINGS USING AN UNSUPERVISED CONVOLUTIONAL NEURAL NETWORK AUTOENCODER

AUTOR:
GABRIEL IGNACIO RUDLOFF BARISON

Tesis de grado presentada por ***Gabriel Ignacio Rudloff Barison****, como requisito parcial para optar al titulo de* ***Ingeniero Civil Electrónico*** *y para optar al grado de* ***Magíster en Ciencias de la Ingeniería Electrónica***
Mención Telecomunicaciones y Procesamiento de Señales*.*

Marcelo Soto Hernández _____
Profesor Guía

Marzo 2023.
Valparaíso, Chile.

Resumen

Una rejilla de Bragg en fibra óptica (FBG, por sus siglas en ingles) es una estructura que se forma modulando el índice de refracción del núcleo de una fibra óptica y que refleja un rango estrecho de longitudes de onda. Midiendo el desplazamiento del perfil espectral, las FBG pueden utilizarse como sensores, ya que este está linealmente relacionado con cambios en factores externos, como la temperatura y la deformación. Un conjunto de FBGs inscritas en una o múltiples fibras puede ser utilizado para crear un sensor cuasi-distribuido. El multiplexado por división de longitud de onda (WDM, por sus siglas en inglés) es una técnica comúnmente utilizada para multiplexar sensores FBG. En este método, a cada sensor FBG se le asigna un canal sin superposición con un ancho de banda específico. Los desplazamientos espectrales de los sensores se miden típicamente utilizando una fuente de banda ancha y determinando el máximo de la reflexión en cada canal. El maximo número de sensores se determina por la relación entre el ancho de banda de la fuente y los anchos de banda de los canales. Técnicas que permiten la superposición espectral, utilizando esquemas de procesamiento de señal más avanzado, pueden superar parcialmente esta limitación asignando varios sensores en cada canal, esto se conoce como WDM con superposición espectral (SWDM, por sus siglas en ingles). La detección de las longitudes de onda de los sensores FBG en el contexto de SWDM es un tarea difícil debido a la superposición de los espectros, lo que hace que técnicas convencionales de detección del máximo no puedan obtener resultados cuando hay demasiada superposición. En la literatura se han propuesto soluciones de Machine Learning, pero estas requieren datos para entrenamiento de alta calidad y gran volumen; lo que puede ser difícil de obtener. Este trabajo propone un modelo que se puede entrenar en un conjunto de datos (pretraining) y luego adaptar de manera no supervisada utilizando solo el espectro de otro conjunto de datos (finetuning), sin necesidad de información sobre las posiciones espectrales. De esta manera, los requisitos para la confección de un conjunto de datos se pueden relajar al no requerir información sobre posiciones espectrales, mediante un proceso de pretraining en datos simulados seguido de un finetuning con datos reales. La propuesta de esta Tesis es un convolutional neural network autoencoder diseñado para minimizar el error de reconstrucción entre espectros observados y reconstruidos. Al hacerlo, se mejora la inferencia de las posiciones espectrales al representar los espectros en una variable latente, a partir de la cual se pueden obtener directamente las posiciones espectrales a través de una relación lineal fija. La capacidad de adaptación del modelo se prueba en simulaciones para casos de dos y tres sensores FBG en serie, mostrando resultados prometedores en comparación con otras técnicas que no utilizan datos no supervisados. La adaptación no supervisada del modelo se prueba cambiando los parámetros de simulación y entrenando de manera no supervisada con los nuevos datos con una relación señal-ruido de 20dB. El modelo demuestra una capacidad de adaptación prometedora tanto para arreglos en serie de dos y tres sensores, superando a otros métodos de la literatura por más de un orden de magnitud en términos de error absoluto medio. El modelo también se evalúa en datos experimentales de un arreglo serial de dos sensores, igualando el rendimiento de un metodo de la literatura con mejor desempeño.

Abstract

A fiber bragg grating (FBG) is a structure that is formed by modulating the refraction index of an optical fiber's core, which reflects a narrow range of wavelengths. By measuring the shift in the spectral profile, FBGs can be used as sensors, as the shift is linearly related to the changes in external factors, such as temperature and strain. An array of FBGs inscribed in one or multiple fibers can be used to create a quasi-distributed sensor. Wavelength division multiplexing (WDM) is a commonly utilized technique for multiplexing FBG sensor arrays. In this method, each FBG sensor is assigned to a non-overlapping channel with a specific bandwidth. The spectral shifts of the sensors are typically measured by utilizing a broadband source and determining the peak from each channel reflection separately. The number of sensors that can be supported by WDM is determined by the ratio of the source bandwidth to the channel bandwidths. Techniques that allow for spectral overlap using complex signal processing can partially overcome this limitation by allocating multiple sensors per channel, referred to as spectrally-overlapped WDM (SWDM). Detecting the wavelengths of FBG sensors in the context of SWDM is a challenging task due to the overlap of the spectra, making conventional peak detection unable to obtain the spectral positions in the case of high overlap. Machine learning solutions have been proposed in the literature, but they require a high-quality, high-volume dataset; which can be hard to obtain. This work proposes a model that can be trained on one dataset (pretraining) and then adapted in an unsupervised manner using only the spectrum from another dataset (finetuning), without requiring information on the spectral positions. In this manner, the requirements on the confection of a dataset can be relaxed by not requiring spectral position information by first pretraining on simulated data and then finetuning on real data. The proposal is a convolutional neural network autoencoder (CNN-AE) designed to minimize the reconstruction error between observed and reconstructed spectra. In doing so, improving the inference of spectral positions by representing the spectra in a latent variable from which the spectral positions can be readily obtained through a fixed linear relation. The model adaptation capability is tested in simulations for two and three FBG sensors in series, showing promising results compared to other techniques that do not use unsupervised data. The unsupervised adaptation of the model is tested by changing the simulation parameters and training in an unsupervised manner with the new data with a signal-to-noise ratio of $20dB$. The model demonstrates promising adaptation capability for both two and three sensor serial arrays, outperforming other methods from the literature by more than an order of magnitude in terms of mean absolute error. The model is also evaluated on experimental data of a two-sensor serial array, matching the performance of the best-performing baseline method.

Contents

Agradecimientos

La realización de esta tesis no habría sido posible sin el apoyo incondicional de mi familia y amigos cercanos, quienes estuvieron presentes en cada etapa de este proceso.

En primer lugar, quiero agradecer a mis padres por su constante apoyo y motivación. Sin su amor y confianza en mí, nunca habría sido capaz de llegar hasta aquí. En especial a mi madre por su dedicación, preocupación y amor incondicional. También agradezco a mi hermana por escucharme y darme ánimos en los peores momentos.

Asimismo, quiero agradecer a mis amigos más cercanos, quienes estuvieron ahí para mí brindándome su apoyo emocional, escuchándome cuando lo necesitaba y dándome fuerzas para continuar. Me gustaría mencionar en particular a Diego Riquelme, mi partner de la U, por nuestras salidas en bici donde compartíamos nuestras frustraciones de tesis, a Vale Espinoza y Vale Ballini (también conocidas como *las vales*) por nuestras salidas sanas (y también por las no tan sanas), en especial a la Vale Ballini por preguntarme religiosamente cómo estaba cuando andaba especialmente desaparecido, a Javier Locier por nuestras salidas a la boca a surfear (cuando lográbamos despertarnos temprano), y a Joaquín Montoya por preguntarme odiosamente cómo iba la tesis cada vez que podía.

Agradezco también al profesor Marcelo Soto por su guía y apoyo durante todo el proceso de investigación. Sus conocimientos y experiencia fueron esenciales para el éxito de este trabajo. También quiero agradecer a mis compañeros del laboratorio, Felipe Diaz y Vinchenzo Saltarini, con quienes senti un apoyo y camaraderia que significo mucho. Rodrigo Fiorin fue tambien un gran apoyo en el desarollo de la parte experimental de esta tesis por el cual estoy muy agradecido.

Parte del financiamiento de esta investigación fue provisto por FONDEQUIP EQM180226 de Conicyt, FONDECYT Regular 1200299 de ANID, FONDECYT Regular 1211980 de ANID, y Basal FB008 de ANID.

Chapter 1

Introduction

Fiber Bragg Grating (FBG) sensors are advanced sensing devices that use periodic changes in the refractive index of optical fibers to reflect certain wavelengths of light and transmit others. These sensors are highly accurate and reliable, able to measure various physical quantities with precision [1]. Depending on the material and construction, they can have a wide temperature range (-200°C to +1200°C), can withstand harsh environments, and are resistant to electromagnetic interference [2]. FBG sensors are small and lightweight, making them easy to install and integrate into various systems. FBGs can be distributed along one or multiple optical fibers to form an array that functions as a quasi-distributed sensor. They are widely used in industries such as aerospace, energy, transportation, and structural health monitoring. The fundamentals of this technology are described in detail in chapter 2. The simulation principles of FBG reflection spectrum are also presented, together with an accurate simulation of the serial topology.

One manner to multiplex an FBG array is through wavelength division multiplexing (WDM). In this multiplexation scheme, each sensor is assigned to non-overlapping wavelength channels and is probed by a broadband source. Then each spectral position is obtained by detecting the peak of the reflected spectrum. An alternative is spectrally overlapped WDM (SWDM), a similar scheme to WDM where each channel contains a fixed set of sensors whose spectra are allowed to overlap. In this case, the peak positions are not straightforward to obtain, and instead, more advanced signal processing techniques should be used. This multiplexation scheme allows using a larger number of sensors with a trade-off in higher complexity of the demultiplexation scheme. SWDM is described in detail in chapter 3 together with the demultiplexation strategies present in the literature.

One solution for SWDM demultiplexation is the use of regression models. These consist of machine learning (ML) models that learn a mapping between an input and an output in a data-driven manner. In this particular case, the input is the spectrum and the output, or target, corresponds to the spectral positions. Artificial neural networks (ANN) are a subset of ML models that are inspired by the structure and function of the human brain. They are composed of layers of interconnected ”neurons”, which process and transmit information. By training an ANN on a large dataset, it can learn to recognize patterns and make intelligent decisions based on that data. This work proposes an ANN-based solution

in the context of SWDM. The necessary background knowledge about ANN is presented in chapter 4.

Given the data-driven nature of ML models, they require a high-quality and high-volume dataset for training. This can be problematic in a setting where data is scarce or difficult to obtain in some scenarios. In the particular case of SWDM, a ground truth of the target is not available and instead should be inferred from an external variable. The creation of a dataset for training should account for the manipulation of a measurand, such as tension or temperature, within the range of operation of each FBG and concurrent indirect measurement of the spectral positions. In this work, we propose to relax this requirement by an initial phase of supervised training on simulated data, termed *pretraining*, and a subsequent phase of unsupervised training, termed *finetuning*. This process is known as unsupervised domain adaptation and refers to adapting a model from one distribution of data to another slightly different only by updating the model with the input data of the new domain.

This work proposes the use of an autoencoder (AE) for this task. AE is a type of ANN structure with two parts, an encoder and a decoder that are connected sequentially and together approximate an identity mapping. AEs are designed to constrain the information passed between encoder and decoder so that it implicitly learns an efficient representation, referred to as a latent variable. The proposal is designed so that the target can be obtained by a fixed linear relation to the latent variable, and that pretraining of the encoder on a simulated dataset and posterior finetuning of the AE over a new dataset improves the inference over the new dataset. To ensure that the latent variable maintains a meaningful relationship with its previous representation after finetuning, certain constraints or biases, known as inductive biases, are introduced into the model. The proposal of this work and its design process is described in detail in chapter 5 together with the evaluation of its capacity under simulated adaptation scenarios as well as early tests with real data.

Chapter 2

Fiber Bragg Grating Fundamentals

2.1 Intuitive Principle

Color can be defined as the visual perceptual property derived from light interacting with photoreceptors in our eyes. What we experience as color is actually light reflecting from objects, and through this process light is modified, granting us with information about the color of the object when it arrives at our retina. The color we perceive is generally[1] determined by the light source shining onto the object and the wavelengths that are reflected in the direction of our eyes.

But what properties do grant an object with a certain color? The most common way is through compounds that absorb certain wavelengths. The observed color is then determined by the remaining light reflected to the eyes. The color of rocks is determined by its mineral composition, for example, cobalt and manganese gives pink, chromium deep green and copper blue or green. Organisms usually achieves color by the use of pigments, molecules specialized in the absorption of certain wavelengths whose production is learned trough the process of evolution in order to achieve competitive advantages such as camouflage or features that draw the attention of the opposite sex. Roses, for example, elicit certain colors to attract their pollinators. Modifications to the gene that generated the pigment can induce changes in the colors it absorbs, some can be seen in figure 2.1.

[1] For simplicity we do not consider optical illusions

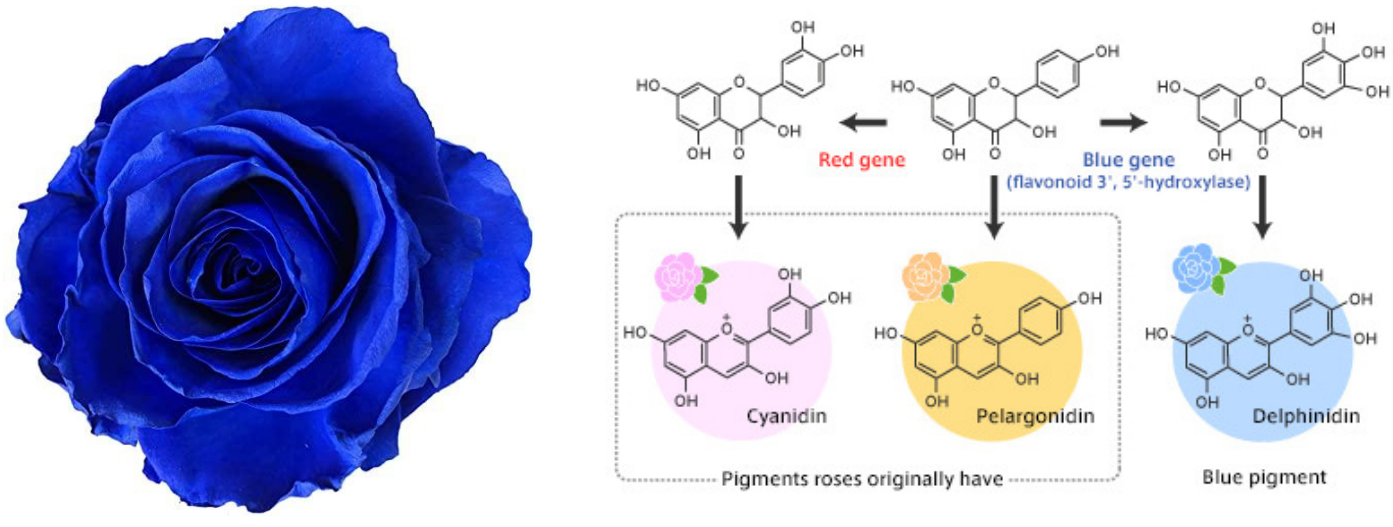


Figure 2.1: A blue Rose (left). Pigments responsible for the color of roses (right).

Some colors such as blue are specially difficult to achieve [3], this has led some species to manage to evoke it in other ways. One interesting example is the Blue Morpho Butterfly (see figure 2.2) that achieves this by the use of structural color, tailored to reflect the blue color. It does this through scales with fine ridges whose cross section looks close to a pine tree, with branches (lamelae) spaced with a periodicity in the order of the reflected wavelength[2]. This structures can be seen in figure 2.2.

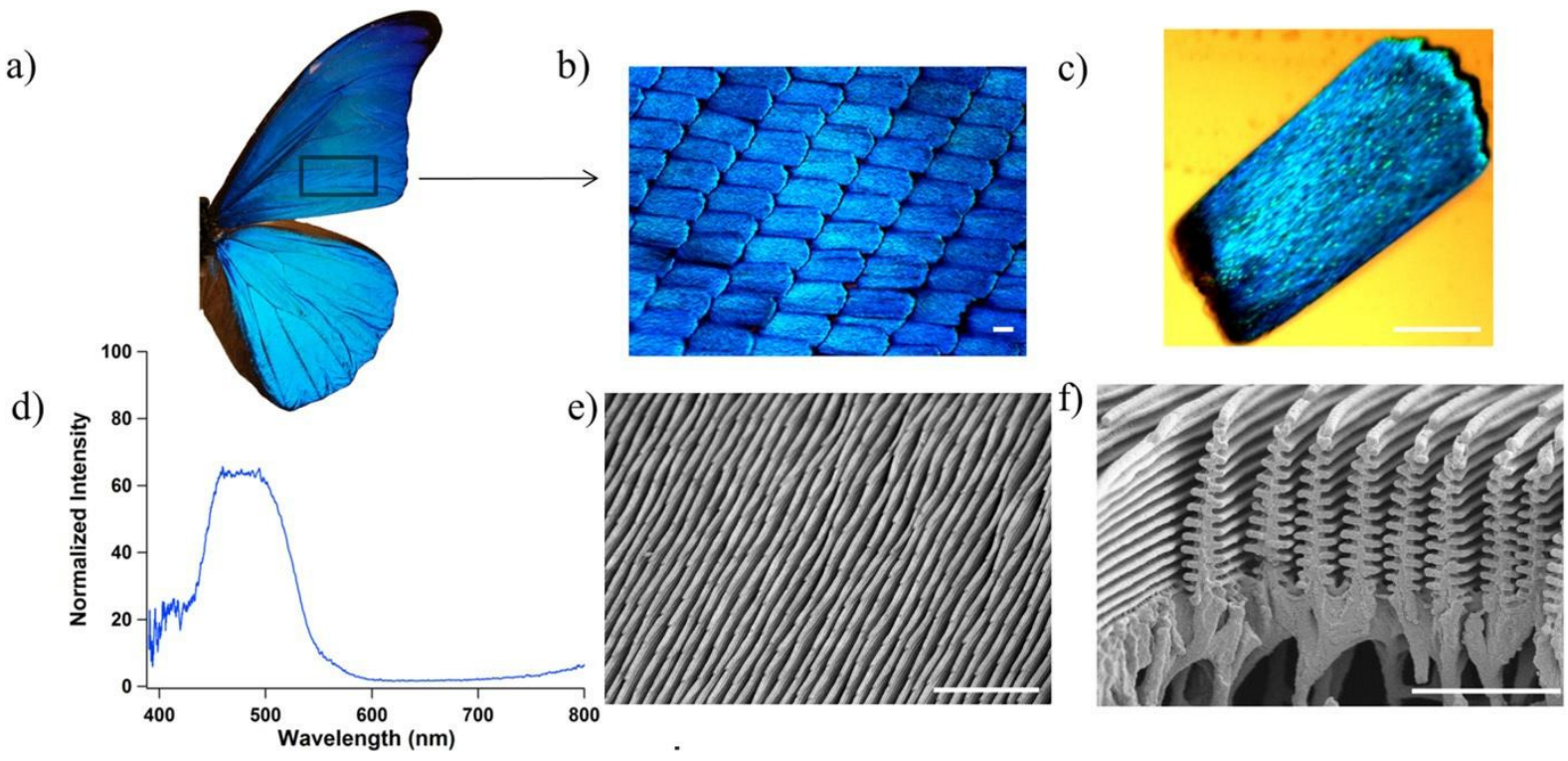


Figure 2.2: The Blue Morpho Butterfly and the structures that grant it with its blue color. Image taken from [5].(a) Blue Morpho's wing. (b) Wing scales. (c) Single scale. (d) Reflection spectrum. (c) Top view of ridges. (d) Side view of ridges showing the periodicity of lamelae.

Some species, particularly of reptiles, take this one step further and use this to dynamically change the colors of their skin. A classic example is the chameleon[3] [6] (see figure

[2]See [4] for an appropriate derivation of the effect.

[3]It is actually a misconception that they employ this as camouflage.

2.3), that has three-dimensional structures of crystals whose distance can be modulated by contraction and relaxation of muscles, and by extension modulate the elicited color. This structures can be seen in figure 2.3.

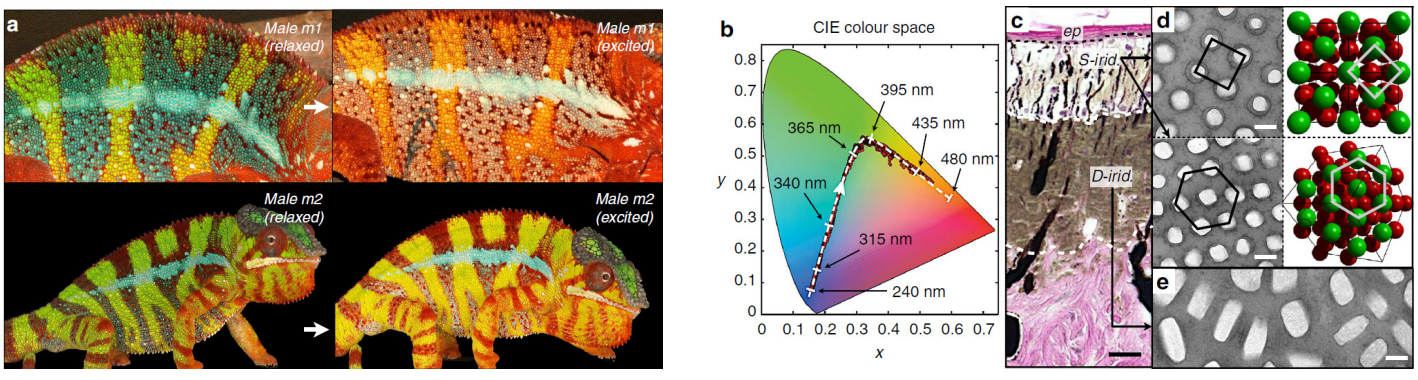


Figure 2.3: (a) Two chameleon displaying their change of color, (b) The range of hue change of chameleons and (c) a cross section of its skin. (d) Images of S-iridophores structures in relaxed and exited states paired with three-dimensional models; and (e) an image of the D-iridophores structure. Taken from [6]

As a proxy for understanding this phenomenon, we can look at the Bragg's law. It states that a crystal lattice with a coherent light shone onto it will elicit peak reflectance for a set of incidence angles, dependent on the spacing between lattice planes. This set of angles is characterized by the Bragg's condition given by $\lambda n = 2d\sin(\theta)$ where λ is the wavelength, $n \in \mathbb{N}$ the diffraction order, d the distance between planes and θ the angle of incidence and reflection of the light[4], this is depicted in figure 2.4. This relation is derived by noting that the path difference that generates an integer number of wavelengths will interfere constructively. This phenomenon is similar to iridescence, where instead of layers of atoms we have a thin film that, when shone with a broadlight source, elicits a color pattern given by colors destructively interfering at determined angles.

[4]Note that even though atoms reemit on every direction, since we assume lattice planes, the added effect is of specular reflection. This conclusion comes from the Huygens–Fresnel principle.

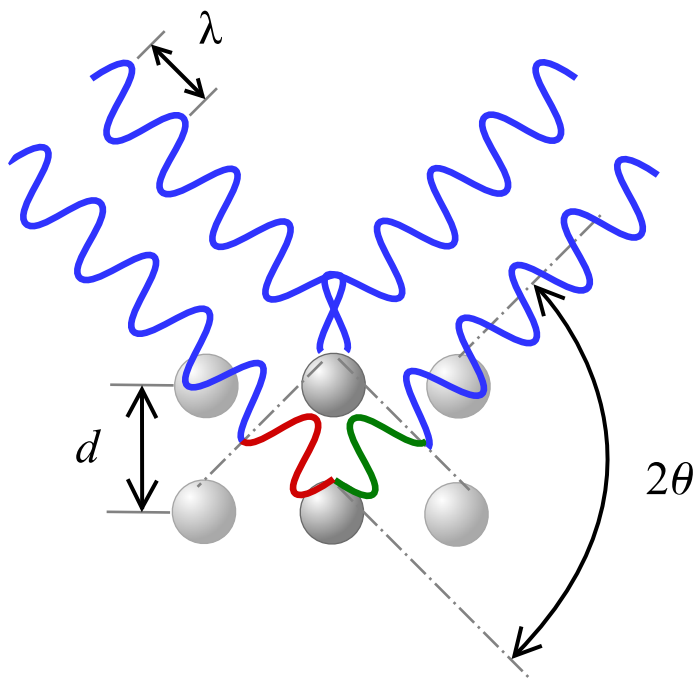


Figure 2.4: Section of a crystal lattice and depiction of specular reflection. Both incidence and reflection angles are θ.

A distributed Bragg reflector (DBR) is a periodic structure comprised of a material with periodic varying effective refractive index that elicits a reflection spectral profile. These are often constructed by alternation of two materials with different refractive index and thickness, which are chosen to tailor a desired photonic stopband. A common choice is the quarter-wave mirror [7] constructed such that a desired wavelength with normal incidence has a $\lambda/4$ phase change in each of the layers, in this manner all reflections interfere constructively with each other.

As an example, consider a DBR constructed from alternating layers of a two dielectrics as depicted in figure 2.5. In particular we choose glass as high refractive index (n_h) medium and air as the low refractive index (n_l) medium, making $n_l \approx n_0$. This will allow us to understand the functioning principle of DBR as well as a simplified version of the blue morpho's scales. Lets first consider the first interface, here A has a π phase change since it is a Fresnel reflection between a low and high refractive index media, B has a π phase change given by the traversal of the layer and no interface phase change since the following is a high to low refractive index interface. On the other hand, C has a 2π phase change given by the two layers it traverses and a π phase change given by the low to high refractive index interface reflection. As can be seen; A, B and C all have the same phase; making their interference constructive, the same is true for D which can be easily checked.

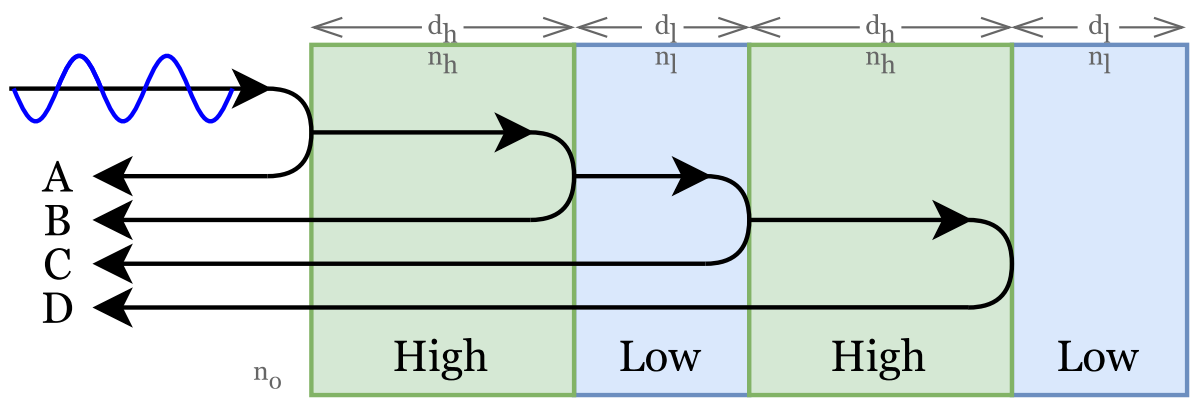


Figure 2.5: DBR with two alternating layers ($N = 2$), with an incoming wavelength and the direct reflections from every interface. Internal multiple reflections are not depicted.

The normal reflectance in each interface is given by $\rho = \left(\frac{n_h - n_l}{n_h + n_l}\right)^2$ [8] and the transmittance is $\tau = 1 - \rho$. Note that the reflactance of a single interface is very small, in the order of 4×10^{-2} (considering $n_h = 1.5$ and $n_l = 1$). The total reflectance is given by

$$P = \rho + \rho\tau^2 + \rho\tau^4 + \rho\tau^6 = \rho \sum_{n=0}^{N+1} \tau^{2n} = \rho \frac{1 - \tau^{2(N+2)}}{1 - \tau^2} \tag{2.1}$$

For our example, where the number of alternated layers is $N = 2$, this structure increases the reflection of a single interface close to three times. If we take a larger number of layers, closer to that of Blue Morpho's lamelae, which are in the order of $N = 10$, we get a reflectivity close to 40%. Note that this results are just an approximation since for simplicity of the example we have not considered the internal multiple reflections. We will see a proper approach to solve this in section 2.4.

2.2 Uniform FBG

FBGs are a subset of DBRs constructed by inscription of a periodic modulation in the core refractive index of an optical fiber. The most basic FBG is the uniform FBG, which consists of a fiber section with a constant sinusoidal modulation of the core refractive index, which can be modeled as $n(z) = n_1 + \delta n(z)$ where n_1 is the unperturbed core refractive index and the perturbation is given by

$$\delta n(z) = \Delta n_{dc} + \Delta n_{ac} \cos\left(\frac{2\pi z}{\Lambda}\right) \tag{2.2}$$

where $z \in (0, L)$ is the axial distance with L the FBG length, Δn_{dc} and Δn_{ac} are the *DC* and *AC* components of the modulation amplitude and Λ is the grating period. The *AC* component is usually related to the *DC* component through $\Delta n_{ac} = \nu \Delta n_{dc}$ [2], [9], where ν is the fringe visibility [5]. This perturbation is visualized in figure 2.6.

[5] We choose not to use this relation, since zero-dc grating manufacturing is possible.

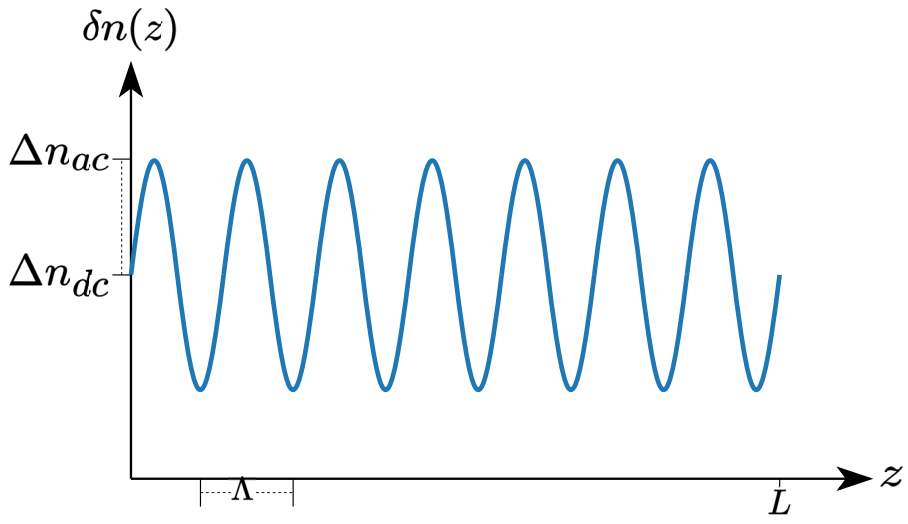


Figure 2.6: Perturbation of the refractive index for a uniform FBG.

This kind of perturbation produces an amplitude reflectivity for single mode operation characterized by [2], [9]:

$$\rho(\lambda) = \frac{-\kappa \sinh(sL)}{\hat{\sigma}\sinh(sL) + is\cosh(sL)} \tag{2.3}$$

where $\hat{\sigma} = \Delta\beta + \sigma$ is the self-coupling coefficient, $\Delta\beta = \beta - \frac{\pi}{\Lambda}$ is the detuning wavevector with $\beta = \frac{2\pi n_{eff}}{\lambda}$ the phase constant, where n_{eff} is the effective refractive index. s is a parameter defined as $s \triangleq \sqrt{\kappa^2 - \hat{\sigma}^2}$ and the *AC* and *DC* coupling coefficient are $\kappa = \frac{\pi}{\lambda}\Delta n_{ac}\eta$ and $\sigma = \frac{2\pi}{\lambda}\Delta n_{dc}\eta$ with η a modal overlap factor [10], which for this case is the power confinement factor. From the amplitude reflectivity we can get the power reflectivity $R = |\rho|^2$ given by

$$R(\lambda) = \frac{\kappa^2 \sinh^2(sL)}{\hat{\sigma}^2\sinh^2(sL) + s^2\cosh^2(sL)} = \frac{\kappa^2 \sinh^2(sL)}{\kappa^2\cosh^2(sL) - \hat{\sigma}^2} \tag{2.4}$$

An example of the power reflection spectral shape of an FBG can be seen in figure 2.7, as can be seen it has a shape similar to a $sinc$ function with a mainlobe and smaller sidelobes surrounding it.

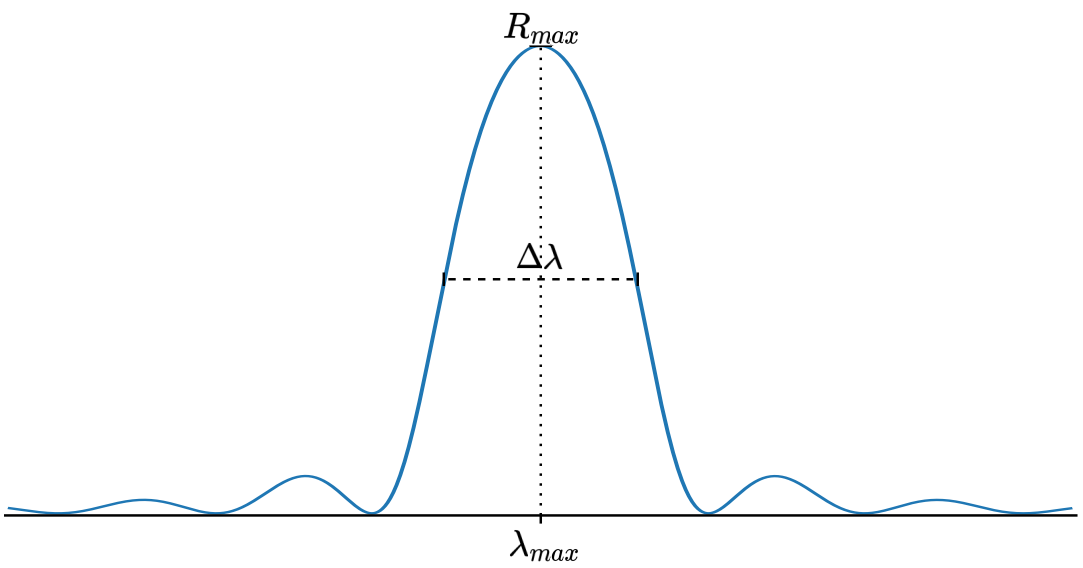


Figure 2.7: Power reflection spectral shape of a uniform FBG with low saturation.

Note that both reflectivity expressions are dependent on the wavelength λ through the detuning wavevector which can be reformulated as $\Delta\beta = 2\pi n_{eff}(\frac{1}{\lambda} - \frac{1}{\lambda_B})$, where λ_B is the Bragg wavelength defined as $\lambda_B \triangleq 2n_{eff}\Lambda$. The reflectivity is maximized when $\hat{\sigma} = 0$ where it takes the value $R_{max} = \tanh^2(\kappa L)$ at the wavelength $\lambda_{max} = \left(1 + \frac{\eta\Delta n_{dc}}{n_{eff}}\right)\lambda_B$ [9]. Note that $\lambda_B = \lambda_{max}$ only when $\Delta n_{dc} = 0$.

The normalized bandwidth as measured between the first two zeros is given by [9]

$$\frac{\Delta\lambda_0}{\lambda} = \frac{\eta\Delta n_{ac}}{n_{eff}}\sqrt{1 + \left(\frac{\lambda_B}{\eta\Delta n_{ac}L}\right)^2} \tag{2.5}$$

There is no analytical characterization for the full-width-at-half-maximum (FWHM) bandwidth $\Delta\lambda$. However, there is an approximation given by Othonos[1] but its deduction is not clear, and in the simulations shown later in this Thesis yielded significant deviations. Othonos's approximation seems to be based in the fact that for high saturation the mainlobe is close to a rectangular function, making $\Delta\lambda \approx \Delta\lambda_0$, while for lower saturation it has a shape close to a Gaussian (see figure 2.7), making $\Delta\lambda < \Delta\lambda_0$. This makes a reasonable approximation to be defined as

$$\Delta\lambda = S(\zeta)\Delta\lambda_0 \tag{2.6}$$

where $S : \mathbb{R}^+ \mapsto]0, 1]$ is a saturation function that maps $\zeta \triangleq \kappa L$ to a proportion value. Othonos uses the approximation $S \approx 1$ for strong gratings and $S \approx 0.5$ for weak gratings. This is a coarse approximation since there is an smooth transition between weak and strong gratings. Instead we choose to approximate this saturation function as described in appendix A.1.

An FBG can have a change in the period of the grating along its length, which is known as chirp. This thesis does not account for chirp because no chirped gratings are used in this work. For more in depth characterization of uniform FBGs and chirped FBGs refer to [2], [9]. A miriad of FBG structures exist, which complexify over the uniform FBG, such as apodized, chirped, phase-shifted, long period gratings (LBG) and superstructure gratings. For characterization on these, refer to [2], [9].

2.3 Coupled Mode Theory

The characterization from the previous section are obtained through coupled mode theory [11], [12]. From this perspective, the perturbation in the core of an FBG generates coupling between modes, which on an unperturbed fiber would not interact. For the case of a uniform FBG, the coupling is between a forward propagating mode and its backward propagating counterpart. In this section, we will outline a comprehensive understanding of applying coupled mode theory to arrive at those results. A derivation of coupled mode theory fundamentals is beyond the scope of this work. For the interested reader refer to [12].

Here we take a more general definition of the refractive index perturbation that includes chirp, which is a change in the grating period along its length. Then the refractive index perturbation is modeled as

$$\delta n(z) = \Delta n_{dc} + \Delta n_{ac} \cos\left(\frac{2\pi z}{\Lambda} + \phi(z)\right) \tag{2.7}$$

where the chirp is quantified as a phase change $\phi(z)$ dependent on the longitudinal position. The results presented here bellow can be obtained for arbitrary spatially periodic nonsinusoidal perturbations by decomposing them as a Fourier expansion [2] and then following analogous deductions.

We can express the electric field as a superposition of forward and backward propagating modes as [9]

$$\vec{E}_t(x, y, z, t) = \sum_j [A_j(z)e^{i\beta_j z} + B_j(z)e^{-i\beta_j z}]\vec{\xi}_{jt}(x, y)e^{-i\omega t} \tag{2.8}$$

where $A_j(z)$ and $B_j(z)$ are the amplitudes of the forward and backward propagating jth mode and $\vec{\xi}_{jt}$ represents the radial transverse field distribution of the jth mode. The exchange of energy between modes is described by the coupled wave equations[6] [9]

[6] Assuming slowly varying amplitudes.

$$\begin{aligned}\frac{dA_j(z)}{dz} = &\quad i\sum_k A_k(z)(K^t_{kj}(z)+K^z_{kj}(z))e^{i(\beta_k-\beta_j)z} \\ &+ i\sum_k B_k(z)(K^t_{kj}(z)-K^z_{kj}(z))e^{-i(\beta_k-\beta_j)z}\end{aligned} \tag{2.9}$$

$$\begin{aligned}\frac{dB_j(z)}{dz} = &- i\sum_k A_k(z)(K^t_{kj}(z)-K^z_{kj}(z))e^{i(\beta_k-\beta_j)z} \\ &- i\sum_k B_k(z)(K^t_{kj}(z)+K_{kj}(z)^z)e^{-i(\beta_k-\beta_j)z}\end{aligned} \tag{2.10}$$

the coupling coefficients $K^t_{kj}(z)$ and $K^z_{kj}(z)$ quantify the energy exchange. The transverse coupling coefficient $K^t_{kj}(z)$ is given by [9]

$$K^t_{kj}(z) = \frac{\omega}{4}\iint \Delta\varepsilon(x,y,z)\vec{\xi}_{kt}\vec{\xi}^*_{jt}dxdy \tag{2.11}$$

where $\Delta\epsilon$ is the perturbation to the permittivity. The longitudinal coefficient $K^z_{kj}(z)$ is generally negligible [9][7]. Assuming the refractive index perturbation is bounded to the core we can describe the transverse coupling coefficient as [9]

$$K^t_{kj}(z) = \frac{\omega}{4}\iint_{core} \Delta\varepsilon(x,y,z)\vec{\xi}_{kt}\vec{\xi}^*_{jt}dxdy \tag{2.12}$$

Using the approximation[8] $\Delta\varepsilon \approx 2n\delta n\varepsilon_0$ valid when $\delta n \ll n$, we have that the transverse coupling can be expressed as

$$K^t_{kj}(z) = \sigma_{kj}(z) + 2\kappa_{kj}(z)\cos\left(\frac{2\pi}{\Lambda}z + \phi(z)\right) \tag{2.13}$$

where the *DC* and *AC* coupling coefficients are defined as

$$\sigma_{kj}(z) = \frac{\omega n_1}{2}\varepsilon_0\Delta n_{dc}(z)\iint_{core}\vec{\xi}_{kt}\vec{\xi}^*_{jt}dxdy \tag{2.14}$$

$$\kappa_{kj}(z) = \frac{\omega n_1}{4}\varepsilon_0\Delta n_{ac}(z)\iint_{core}\vec{\xi}_{kt}\vec{\xi}^*_{jt}dxdy \tag{2.15}$$

For the single mode case, where a forward mode with amplitude $A(z)$ couples with a backward mode with amplitude $B(z)$ the coupled mode equations can be rewritten as [9]

$$\frac{dR(z)}{dz} = i\hat{\sigma}(z)R(z) + i\kappa(z)S(z) \tag{2.16}$$

[7]The expressions for the longitudinal coefficients can be found in [12]

[8]This approximation is obtained from $\Delta\varepsilon = \epsilon_0\Delta(n(z)^2) \approx 2\varepsilon_0 n\delta n(z)$ with $\varepsilon = \varepsilon_r\varepsilon_0$ and $n^2 = \varepsilon_r$

$$-\frac{dS(z)}{dz} = i\hat{\sigma(z)}S(z) + i\kappa(z)^* R(z) \tag{2.17}$$

where a synchronous transformation was used, characterized by the substitution $R(z) = A(z)e^{i\Delta\beta z - \phi/2}$ and $S(z) = B(z)e^{-i\Delta\beta z + \phi/2}$. The *AC* coupling coefficient is $\kappa(z) = \kappa_{ij}(z)|_{i=j}$, the *DC* coupling coefficient is $\sigma(z) = \sigma_{ij}(z)|_{i=j}$ and the *DC* self-coupling is defined as

$$\hat{\sigma}(z) \triangleq \Delta\beta + \sigma(x) - \frac{1}{2}\frac{d\phi(z)}{dz} \tag{2.18}$$

Arbitrarily complex FBG structures can be characterized through coupled-mode theory by solving the coupled-mode equations 2.16 and 2.17 through numerical methods, such as the fourth order Runge-Kutta method [13]. Analytical expressions exist only for the uniform case as characterized in section 2.2. In the following section we will present an approach to obtain analytical approximations beyond the uniform profile.

There are grating structures tailored to couple modes beyond the forward fundamental mode and its backward counterpart, for example, LPG couple core modes and cladding modes. An analogous approach must be followed from the coupled mode equations 2.9 and 2.10. For examples on these refer to [9], [13].

2.4 Transfer-Matrix Method

The Transfer-Matrix method is a computational method used to calculate electromagnetic waves interaction with stratified medium [8]. In this context, it consists in approximating an arbitrary FBG structure as a piece-wise uniform FBG, each section can then be fully characterized through a transfer matrix defined as [9]:

$$T_j = \begin{pmatrix} \cosh(sl_j) - i\frac{\hat{\sigma}_j}{s}\sinh(sl_j) & -i\frac{\kappa_j}{s}\sinh(sl_j) \\ i\frac{\kappa_j}{s}\sinh(sl_j) & \cosh(sl_j) + i\frac{\hat{\sigma}_j}{s}\sinh(sl_j) \end{pmatrix} \tag{2.19}$$

where the coefficients with subscript j are the coefficients defined in section 2.2 of the j-th section. The counter propagating fields at each interface are related as

$$\begin{bmatrix} a_j \\ b_j \end{bmatrix} = T_j \begin{bmatrix} a_{j-1} \\ b_{j-1} \end{bmatrix} \tag{2.20}$$

as depicted in figure 2.8. Then a piece-wise uniform FBG can be fully described as

$$\begin{bmatrix} a_J \\ b_J \end{bmatrix} = T \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} \tag{2.21}$$

$$T = \begin{pmatrix} T_{00} & T_{01} \\ T_{10} & T_{11} \end{pmatrix} = T_J T_{J-1} \cdots T_1 = \prod_{j=J}^{1} T_j \tag{2.22}$$

and the field reflectivity can be characterized as $\rho = b_0/a_0 = T_{10}/T_{11}$ by assuming $b_J = 0$. A piece-wise approximation is depicted for an apodized FBG in figure 2.8 where $J = 5$.

An apodized FBG has a spatial modulation of the grating with a peak at the center and with decreased amplitude to the sides. This has a sidelobe suppression effect over the reflection spectral shape which is desired in most applications.

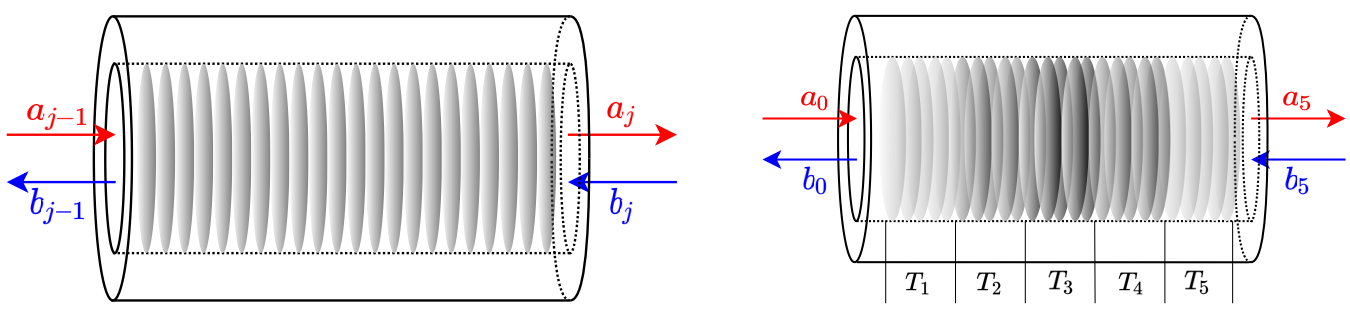


Figure 2.8: Uniform FBG (left). Apodized FBG discretized as a series of 5 uniform FBGs with varying modulation levels (right).

Another interesting FBG, which we can characterize in this way, is the phase-shifted FBG [9]. This is an FBG with a discrete gap in the grating that has the effect of adding a bandpass notch inside the stopband of the FBG. This is modeled as a phase shift matrix of the form

$$F_\phi = \begin{pmatrix} \mathrm{e}^{-\frac{i\phi}{2}} & 0 \\ 0 & \mathrm{e}^{\frac{i\phi}{2}} \end{pmatrix} \tag{2.23}$$

where the phase change of a gap of length Δz is given by [9]

$$\frac{\phi}{2} = \frac{2\pi n_{eff}}{\lambda}\Delta z \tag{2.24}$$

We can also characterize a lossy section, where the field power is scaled by a constant as $P_{out} = \alpha P_{in}$ with $\alpha \in [0, 1]$. The equivalent matrix form that relates the fields of such a section, refered as transmissivity matrix, is given by

$$F_\alpha = \begin{pmatrix} \sqrt{\alpha} & 0 \\ 0 & \sqrt{1/\alpha} \end{pmatrix} \tag{2.25}$$

2.5 Fabrication

The first demonstration of FBG inscription, known as *internal inscription* [1], consists in using a narrow source on a fiber. The incoming light in the fiber interferes with the reflected light at the end of it creating an standing wave interference pattern. The high intensity peaks start generating fringes in the refractive index of the core until an FBG is formed. The periodicity of the grating is given by the standing wave wavelength $\Lambda = \lambda_w/2$, where λ_w is the source wavelength.

[9] Note that this only holds true for gaps below the coherent length, we will address larger gaps in section 2.9.1

Inscriptions are possible thanks to photosensitivity, a phenomenon by which a material changes its refractive index or opacity permanently due to exposure to certain wavelengths. The photosensitivity to ultraviolet (UV) is usually increased for the purpose of FBG manufacturing through doping with rare earths or through hydrogen loading [2]. Then, UV light shone onto it will produce permanent local increases of the refractive index proportionally to the intensity and the exposure time.

Internal inscription is substantially limited as only weak gratings can be inscribed and the possible fringe periods are dependent on the photosensitive range of wavelengths. Furthermore, it requires a variable wavelength source to change the inscribed fringe period. An alternative to improve this is *external inscription* (EI), also known as *interferometric inscription*, in which an interference pattern is focused onto the core of the photosensitive fiber from its side.

Amplitude-splitting inscription [1] is one type of EI, in which a beam is split in two and each beam is redirected with mirrors to arrive at the inscription point with an angle between them. Cylindrical lenses are used to focus each beam into the core and to produce an interference pattern in it. This inscription method requires a source with high coherence length and for the paths of the split beams to have phase changes as similar as possible. An example of this scheme is shown in figure 2.9. The period of the inscribed fringe is not dependent solely on the source wavelength λ_w but can also be controlled through the beam angle ϕ as

$$\Lambda = \frac{\lambda_w}{2 \sin \varphi} \tag{2.26}$$

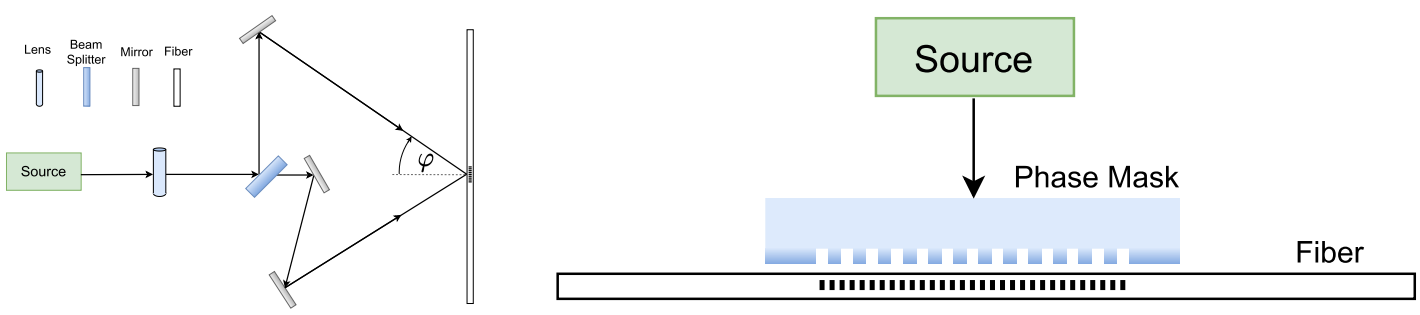


Figure 2.9: Amplitude Splitting interferometer (left). Phase-mask Interferometer (right). The size of the elements are exagerated for ilustration purposes.

Another similar approach is through the use of *wavefront-splitting interferometers* [1]. In this approach a source is spatially split by reflecting half of the wavefront, in order to have an angle with respect to the other half of the wavefront, which produces an interference pattern between them. The reflection can be obtained either with a mirror perpendicular to the fiber or with a prism that achieves the same effect. The fringe period can be tailored through the angle of the incoming beam.

One of the most used alternatives is *phase-mask inscription* [1], [2] in which a *photo-mask* is used to produce the interference pattern. A photo-mask is a diffractive optical

element that works in a similar way to the classical double slit experiment. A photo-mask can be constructed from a silica glass slab with a rectangular periodic corrugation etched in one of its sides as depicted in figure 2.9. The corrugation pattern can be tailored to generate complex fringe patterns, such as chirps and apodizations with great precision.

Sequential writting is possible through the *point-by-point* technique in which the fringes are inscribed one by one by a beam focused onto the core of the fiber. The source is controlled in an on-off fashion by a slit and the fiber is longitudinally translated by a precise motorized setup. This setup has greater flexibility and allows the inscription of more complex structures such as spatial mode converters, polarization mode converters and rocking filters [1]. This flexibility comes at a trade-off with grating spacing precision due to susceptibility to ambient noise and temperature, allowing only short fringes to be inscribed. More recently femtosecond lasers have been used to manufacture FBGs showing greater thermal stability and flexibility to inscription in different fiber types, even on non-doped fiber [14]. See [2] for an in-depth review of manufacturing processes.

2.6 Sensing Principle

As described in section 2.1, chameleons are able to modulate the elicited color of their skin by means of contraction of the crystal arrays inside their skin. Similarly, FBG spectra can be subject to wavelength shift by means of actuation upon them. In particular any change over the refractive index or the grating period will have a shift effect over the FBG spectrum, and because of this they have great sensitivity to temperature and axial strain.

FBGs are used in the context of optical communication system (refer to [10] for more detail on applications in this area), but here the stability of the reflection spectrum is desirable, making the susceptibility to its environment a problem. Because of this, FBGs used in this context must be adequately isolated from such perturbations.

The susceptibility to its environment makes FBGs an excellent candidate for sensing. The sensitivity to temperature and strain can be understood through the fiber elastic, elastooptic and thermooptic properties [10]. The strain response is linear while the temperature is slightly nonlinear for high temperatures [10]. This relation can be expressed as a shift of the Bragg wavelength $\Delta\lambda_B$ linearly dependent on the axial strain change[10] $\Delta\varepsilon_z$ and temperature change ΔT as

$$\Delta\lambda_B = \Psi_{\varepsilon_z}\Delta\varepsilon_z + \Psi_T\Delta T \tag{2.27}$$

where Ψ_{ε_z} and Ψ_T are the axial strain and temperature sensitivities. Sensitivities are in the order of $1[pm/\mu\varepsilon]$ for Ψ_{ε_z} and in the order of $10[pm/K]$ for Ψ_T [1], [10].

Beyond temperature and strain, there is sensitivity to hydrostatic pressure P and to changes in the cladding refractive index n_2 [2]. Further sensing applications can be achieved

[10]The axial strain is defined as $\varepsilon_z = \frac{\delta l}{l}$

by using appropriate coating that transduce the measurand to strain or temperature [10].

To deduce a sensitivity to a certain measurand X(e.g., temperature or strain), the functional dependence to λ_B can be obtained through its effect over n_{eff} and Λ by derivation of the expression $\lambda_B = 2n_{eff}\Lambda$ as

$$\frac{d\lambda_B}{dX} = 2\Lambda\frac{dn_{eff}}{dX} + 2n_{eff}\frac{d\Lambda}{dX} \tag{2.28}$$

The relation between $\Delta\lambda_B$ and ΔX can be expressed as [2]

$$\Delta\lambda_B = \frac{d\lambda_B}{dX}\Delta X = \lambda_B(\alpha_n + \alpha_\Lambda)\Delta X \tag{2.29}$$

where the normalized sensitivities for refractive index and grating period are given by $\alpha_n = \frac{1}{n_{eff}}\frac{dn_{eff}}{dX}$ and $\alpha_\Lambda = \frac{1}{\Lambda}\frac{d\Lambda}{dX}$. For the case of strain, the sensitivity is given by [1], [10]

$$\Psi_\varepsilon = \lambda_B(1 - \rho_a) \tag{2.30}$$

with the stress-optic coefficient given by

$$\rho_a = \frac{n_{eff}^2}{2}[\rho_{12} - \nu(\rho_{11} + \rho_{12})] \tag{2.31}$$

where ρ_{12} and ρ_{11} are coefficients of the stress-optic tensor and ν[11] is the Poisson's ratio. For the case of temperature, the sensitivity is given by

$$\Psi_T = \lambda_B(\alpha_n + \alpha_\Lambda) \tag{2.32}$$

Here the coefficients have the same meaning as defined previously, but in this context α_Λ and α_n take the names thermal expansion coefficient and thermo-optic coefficient.

Although analytical expressions for the sensitivities exist, exact parameters for a given fiber are not readily available. Because of this, the values for the sensitivities are usually obtained through calibration.

There is no direct way of decoupling the measurement of a single FBG into the components attributed to each measurands. But, in some cases, it is possible to insulate the FBG in order to make it insensible to one of the measurands. For example, with an insulator coating we can neglect the effect of temperature change. The other alternative is to have m FBGs in the same location, this allows to construct a sensitivity matrix Ψ that relates the spectral shifts $\Delta\lambda_B = \{\Delta\lambda_{B_i}\}_{i=\{1,\cdots,m\}}$ and the measurands change ΔX like $\Delta\lambda_B = \Psi\Delta X$. The measurands can be obtained from the spectral shifts by inverting Ψ, in order to be invertible it must be rank M, this is achieved if the FBGs have different relations between their sensitivities. For more details refer to [1], [2].

[11] Note this is not the same as the visibility defined in section 2.2

2.7 Interrogation

The great sensitivity, immunity to electromagnetic interference, low insertion loss and wavelength encoding make FBGs ideal to be used as sensors. Interrogation can be made with an optical spectrum analyzer (OSA) by measuring the peak reflectivity wavelength shift. For a review on peak tracking techniques from OSA readings of an FBG refer to [15].

In practical applications, this option is often not sufficiently compact or economic. An alternative is the edge filter. In this scheme, a filter with a linear transition band is used in order to characterize the variation in the Bragg wavelength with the variation in intensity (see figure 2.10). There is a trade-off between the resolution and the dynamic range by means of variation in the cutoff slope. Another option is the tunable filter, in which a tuneable filter is used in a closed loop control system by locking it to the Bragg wavelength (see figure 2.10).

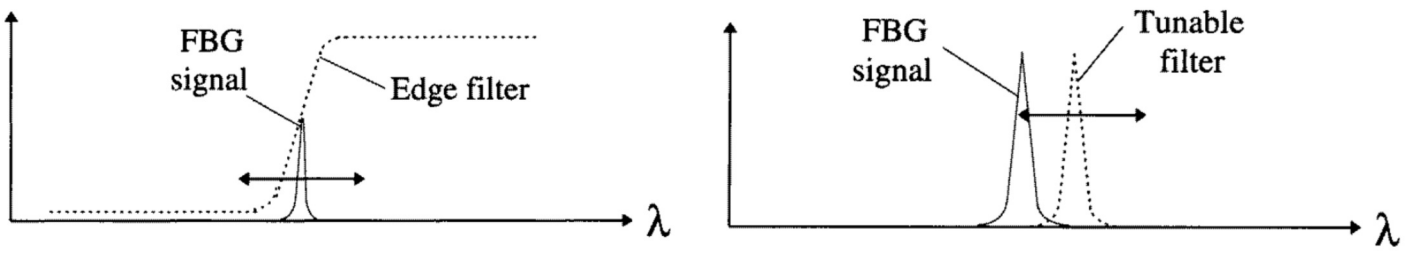


Figure 2.10: Principle of the edge filter method (left). Principle of the tunable filter method (right). Taken from [16]

Another option is the interferometric scanning scheme. Here the change in Bragg wavelength is characterized by a change in phase of the interferometer's output. The operational range is characterized by the free spectral range (FSR), which is the period of the phase with respect to the wavelength. This scheme can be extended with two levels of resolution to increase the operational range without a compromise in resolution.

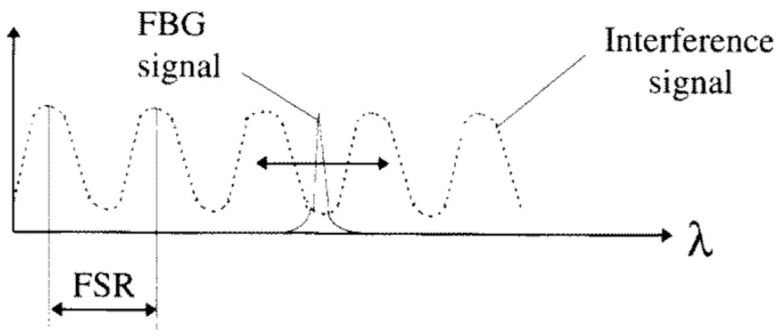


Figure 2.11: Principle of the interferometric scanning method. Taken from [16]

For more details on these interrogation schemes refer to [16]. Inter-FBG measurements are also possible, for details on this, together with more point-like interrogation schemes, refer to [17].

2.8 Quasi-distributed Sensing

A quasi-distributed sensor can be constructed by using multiple FBGs. In this context, each FBG is used as a point-like sensor and they are distributed spatially to have a discrete measurement of a line, a manifold or a volume. In this section, we will outline the alternatives for the topologies and multiplexation schemes used for FBG arrays.

2.8.1 Topology

FBG arrays can be connected in different ways, this is known in this context as topology. The basic two topologies are serial and parallel, depicted in figure 2.12 and figure 2.12, respectively. Each topology has different tradeoffs in terms of used components and crosstalk between sensors, these will be discussed in more detail in the following section. More complex topologies can be constructed as compositions of these basic topologies. An example can be seen in figure 2.12. For more examples refer to [18].

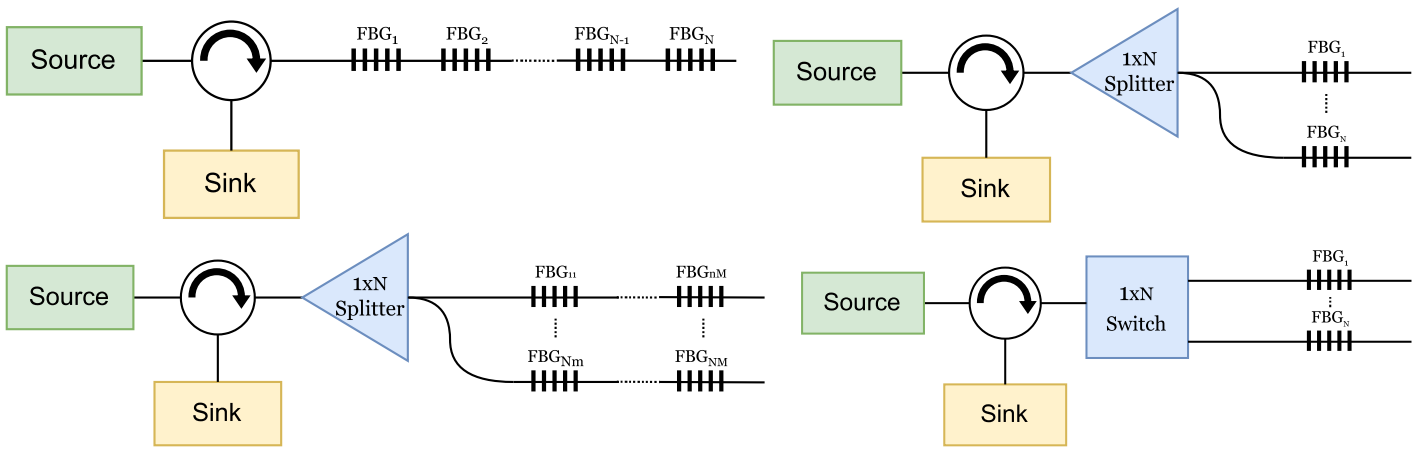


Figure 2.12: Serial FBG array topology (top left). Parallel FBG array topology (top right). Parallel-Serial FBG array topology (bottom left). Example of Spatial Division Multiplexing in a parallel topology (bottom right).

2.8.2 Multiplexation

An FBG array can be interrogated as described in section 2.7, but the signals from each sensor must be separated to obtain the individual wavelength shifts. The means by which the separation is performed is known as multiplexation.

The more straightforward multiplexation scheme is Wavelength Division Multiplexing (WDM). In this scheme, a serial FBG array has each sensor tailored to assigned non-overlapping channels with bandwidth $\Delta\lambda_{B_j}$ given by the expected maximum displacement or by the full dynamic range of each FBG. Then, the spectral shifts are inferred from each channel. Between adjacent channels there is a guard band (GB) allocated to evade crosstalk. A depiction of the WDM spectra can be seen in figure 2.13.

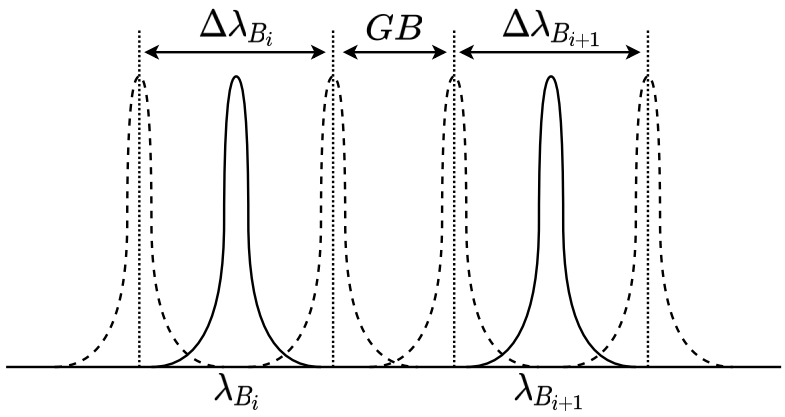


Figure 2.13: Spectrum of two FBG channels from a WDM FBG array. λ_{B_j} are instances of Bragg wavelengths from the j-th FBG. $\Delta\lambda_{B_j}$ is the bandwidth of the j-th channel.

The number of sensors Q supported by this scheme is proportional to the ratio between the broadband source bandwidth and the channel bandwidths plus the guard bands (GB), satisfying the following equation

$$\sum_{i=1}^{Q} \Delta\lambda_{B_i} + (Q-1)GB = \Delta\lambda_s \tag{2.33}$$

where $\Delta\lambda_s$ is the bandwidth of the source and $\Delta\lambda_{B_i}$ is the bandwidth of the i-th channel. If each sensor is allotted an equal channel bandwidth $\Delta\lambda_B$, then the maximum number of sensors is given by

$$Q = \frac{\Delta\lambda_s + GB}{\Delta\lambda_B + GB} \tag{2.34}$$

For more details refer to [19].

Time Division Multiplexing (TDM) is another Multiplexation scheme. This is based on the difference in optical path induced by the spacing of multiple FBGs. When a light pulse is sent through them, the TDM-multiplexed array reflects a train of pulses with different delays. The spectral shift can then be extracted from each of the reflected pulses sequentially. In this case, all FBGs can share the same Bragg wavelength, but they must have very low reflectivity. The effective reflectivity of the main reflection from the i-th FBG is given by

$$\hat{R}_i(\lambda) = R_i(\lambda) \prod_{j=1}^{i-1} T_j(\lambda) \tag{2.35}$$

Where $R_i(\lambda)$ is the reflectivity of the i-th FBG and $T_j(\lambda) = 1 - R_j(\lambda)$ is the transmittance of the j-th FBG. The transmittance terms produce a type of crosstalk called spectral shadowing, which is produced by the modification of the source spectrum by means of the reflected spectrum on downstream FBGs. In addition to the main reflection of each FBG, there will be multiple reflections between different FBGs that will arrive in the same window slot allotted to one FBG. This is known as multiple-reflection crosstalk. Generally first-order reflection are relevant, while higher-order crosstalk is negligible for small reflection FBGs. An example of the different types of reflections is depicted in figure

2.14. If in this case the distance between the sensors is the same, the path *(b)* and *(c)* would arrive simultaneously at the fiber input,generating crosstalk.

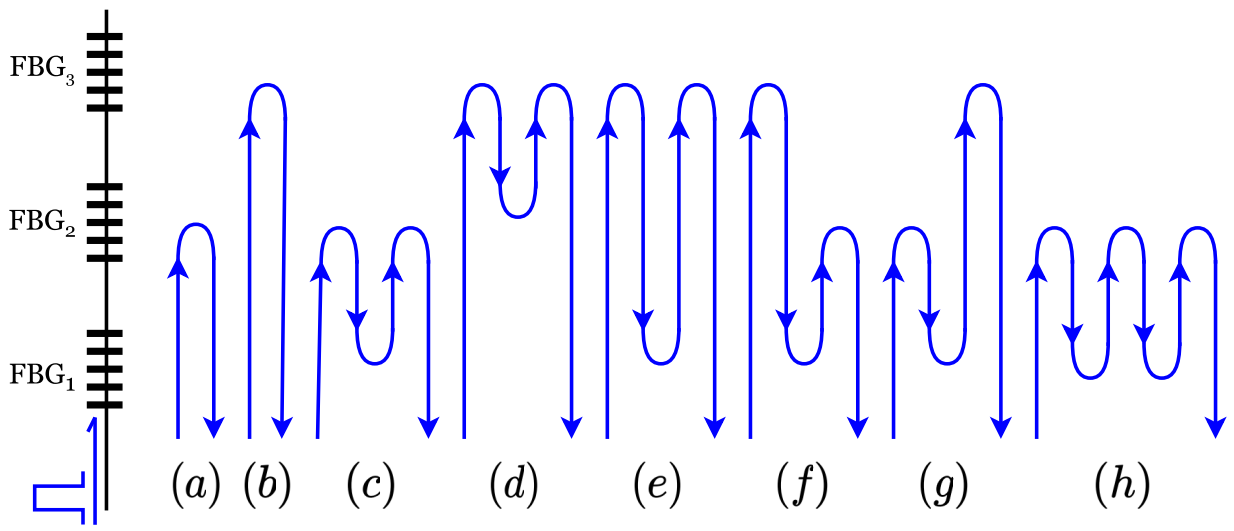


Figure 2.14: Paths in a TDM serial FBG array. (a) and (b) are the single reflection paths from FGBs 2 and 3. (c) to (g) are examples of first-order multi-reflection paths. (h) is an example of higher-order multi-reflection path

A TDM array must meet the following design requirements. The reflectivity from each FBG must be sufficiently small for the reflection from the last FBG to have enough incoming light, leading to an adequate SNR level on arrival to the detector. The width of the pulse Δt must be shorter than the time taken for the light to traverse the distance between adjacent FBGs twice to ensure that the main reflection pulses do not overlap. This is given by

$$\Delta t < \max_{i\in[2,\ldots,Q]} \Delta T_i \tag{2.36}$$

where ΔT_i is the time light takes to traverse a roundtrip in the i-th gap, given by $\Delta T_i = 2\Delta l_i \frac{n_{eff}}{c}$, and Δl_i is the distance of the gap. On the other hand the period between pulses T must be greater than the time of the Q-th reflection roundtrip. This is given by

$$T > \sum_{i=1}^{Q} \Delta T_i \tag{2.37}$$

TDM-based FBG arrays can be constructed with large Q values but they have some limitations. These are mainly *(i)* in the time resolution, limited by the need to wait for all the reflections to arrive at the reciever before sending another pulse, *(ii)* in the spatial resolution, limited by the need for sufficient spatial distance to avoid reflection overlapping, and *(iii)* in the wavelength shift resolution, limited by the pulse length. The crosstalk can be ameliorated substantially by the use of couplers and parallel lines for each FBG or by compensating the spectral shadow with the information of downstream spectra [20]. Another improvement can be obtained by the use of code division multiple access (CDMA) [21], which allows for a substantial reduction in the sampling period by sending pseudorandom bit sequences.In this case, each instance is called a ”chip” and the spectrum of each FBG can be obtained by correlation with previously sent chips. The number of FBGs that can be addressed by chips of length C is given by $Q = 2^C - 1$.

Another multiplexation option is Spatial Division Multiplexing (SDM), where each FBG is in different optical lines, which are interrogated sequentially by means of an optical switch. This topology can be seen in figure 2.12. Further increases in capacity can be made by using composite multiplexation, where two or more multiplexation schemes are used.

2.9 Simulation

Simulation of arbitrary FBG structures is possible either by solving the coupled wave equations 2.16-2.17 through numerical methods, or by using the transfer matrix method and approximating an FBG as a concatenation of uniform FBG with varying characteristics. In this work we focus just on the uniform case with no chirp, so the characterization from section 2.2 is sufficient.

For simulation purposes, we parameterize an uniform FBG by its Bragg wavelength λ_B, by its peak reflectivity R_{max}, its FWHM wavelength $\Delta\lambda$ and by its core effective DC refractive index change Δn_{dc}. To simplify our analysis we choose to set $\Delta n_{dc} = 0$ since in this way there is no offset between λ_B and λ_{max}. We must also set some characteristics of the fiber, such as core and cladding refractive indexes and core radius, by typical values in order to obtain the values of n_{eff} and η. The values used in this work are specified later in section 5.1.1.

From R_{max} we can obtain ζ as $\zeta = arctanh(\sqrt{R_{max}})$. The core effective *AC* refractive index change Δn_{ac} for a given $\Delta\lambda$ can be obtained from equation 2.6 by using the approximation $\lambda \approx \lambda_B \approx \lambda_0$, where λ_0 is the wavelength at the center of the simulation range. It can be shown that, given this assumption, the *AC* refractive index change is given by

$$\Delta n_{ac} = \frac{n_{eff}\Delta\lambda}{\lambda_0\eta}\frac{1}{S(\zeta)}\left(1+\left(\frac{\pi}{\zeta}\right)^2\right)^{-1/2} \tag{2.38}$$

with this, we can compute the values of κ, $\hat{\sigma}$, L and s. From this we can compute the value of the power spectrum $R(\lambda)$ for a given λ. We then compute the power spectrum for N equidistant points in the range $\lambda \in [\lambda_0 - \Delta, \lambda_0 + \Delta]$, and with this we construct a spectrum vector as $r = \{R(\lambda_1), \cdots, R(\lambda_N)\}$.

If instead an apodized FBG is simulated, with sufficient sidelobe suppression, it can be adequately approximated by a Gaussian function [22] given by

$$R(\lambda) = R_{max}\exp\left(-4ln(2)\left[\frac{\lambda-\lambda_B}{\Delta\lambda}\right]^2\right) \tag{2.39}$$

and the spectrum vector r can be constructed analogously.

Another approach for creating synthetic individual spectra is what we term pseudo-simulation. This method involves measuring the spectral profile of a real FBG sensor, that is then shifted according to the spectral position of the simulated instance.

2.9.1 Arrays

In the previous section, we described how to simulate the reflection spectra of a single sensor. In this section, we explore how to simulate an array of sensors in two topologies, namely, parallel and serial topologies. The results of this section could be extended to any topology by following analogous deductions.

Simulating a parallel array is straightforward, as there is no interaction between different sensors, so their reflectivity are just added together as

$$x = \sum_{i}^{Q} a_i r_i = A \cdot R \tag{2.40}$$

where $R = \{r_i\}_{i=\{1,\cdots,Q\}}$ is the $Q \times N$ spectrum matrix with r_i the spectrum vector of the i-th FBG, and $A = \{a_i\}_{i=\{1,\cdots,Q\}}$ is the $Q \times 1$ transmissivity vector with a_i the i-th transmissivity, that accounts for round trip attenuation and the power splitting proportions between the parallel lines given by the optical splitter. Because of this, it must be ensured that $sum(A) \leq 1$.

On the contrary, the simulation of a serial topology is not so straightforward, as there are resonances inside of each cavity, as can be seen in figure 2.14. One naive approach is to use a recursive approach that joins the effect of sensors in a serial manner by incorporating the resonance between each pair of sensors.

To understand this, lets look at the case for just a pair of sensors. The total reflection will be given by the first sensor reflection plus all the infinite reflections on the second sensor, given by the cavity created between the two sensors. This is expressed mathematically as

$$x = r_1 + (1 - r_1)^2 r_2 \sum_{j=0}^{\infty} (r_1 r_2)^j \tag{2.41}$$

We can notice that the last summation term is a geometric series (with $r_1, r_2 < 1$), making the expression equivalent to

$$x = r_1 + (1 - r_1)^2 \frac{r_2}{1 - r_1 r_2} \tag{2.42}$$

By extending this idea for a serial array of Q sensors and adding the transmissivity of the

fiber in between sensors, we can simulate the joint reflectivity $x = \tilde{r}_Q$ by iterating over

$$\tilde{r}_i = \tilde{r}_{i-1} + (1 - \tilde{r}_{i-1})^2 \frac{a_i r_i}{1 - \tilde{r}_{i-1} a_i r_i} \tag{2.43}$$

for $i \in [1, ..., Q]$ with $\tilde{r}_0 = 0$. Although a good approximation, this naive approach does not achieve perfectly accurate results, as the reflectivity characterization assumes that the incoming field after the sensor is null, as we saw in section 2.4. A similar approach to this is proposed in [23], but consists in neglecting higher order reflections.

To make a rigorous simulation of the serial array we must use the Transfer-Matrix Formalism. The reference [24] follows this approach, but fails to take into consideration the gap between sensors, essentially simulating a concatenation of FBGs and neglecting in this way any cavity effects. As we saw in section 2.4, we can simulate a gap as a phase change, characterized by a phase shift matrix F_ϕ, but this assumes perfectly coherent sources and in consequence gaps beyond the coherence length of the source are not well characterized in this way.

To simulate incoherent gaps we turn to thin film physics where there is literature regarding this [25]. To obtain the true reflection spectrum, the result must be integrated over the possible phase changes of each gap. This can be approximated to the desired level of accuracy by taking an adequate number of random phase changes for each gap and averaging the result [26], [27]. This approach is computationally expensive and with a growing expense with number of sensors. One way to alleviate this computation is by sampling an equidistant set of phases $\phi_i \in [0, 2\pi]$ for each gap as proposed in [28].

An approximation that is adequately close to the true value, but with negligible computational cost, consists in constructing power transfer matrices from the amplitude transfer matrices [25], [27]. Given a j-th layer with complex amplitude reflectance ρ_j and transmittance τ_j, its amplitude transfer matrix can be expressed as [25]

$$\mathbb{T}_j = \frac{1}{\tau_j} \begin{pmatrix} 1 & -\rho_j \\ \rho_j & \tau_j^2 - \rho_j^2 \end{pmatrix} \tag{2.44}$$

This expression is related to the transfer matrix defined in section 2.4 by $\mathbb{T}_j = T_j^{-1}$. This means that it expresses the fields on the left of the section in terms of the fields on the right. An approximation for the incoherent layer can be made by constructing a power transfer matrix $\mathcal{T}_j$ by replacing the terms in $\mathbb{T}_j$ by their square amplitudes as [25]

$$\mathcal{T}_j = \frac{1}{t_j} \begin{pmatrix} 1 & -r_j \\ r_j & t_j^2 - r_j^2 \end{pmatrix} \tag{2.45}$$

Using this result to express the total power transfer matrix $\mathcal{T}$ of the serial FBG array and

considering attenuation, is then given by

$$\mathcal{T} = \begin{pmatrix} \mathcal{T}_{00} & \mathcal{T}_{01} \\ \mathcal{T}_{10} & \mathcal{T}_{11} \end{pmatrix} = \prod_{j=1}^{Q} \mathcal{A}_j \mathcal{T}_j \qquad \mathcal{A}_j = \begin{pmatrix} \sqrt{1/a_j} & 0 \\ 0 & \sqrt{a_j} \end{pmatrix} \tag{2.46}$$

where $\mathcal{A}_j$ is the power transmissivity transfer matrix. Note that $\mathcal{A}_j$ is related to the transmissivity matrix defined in section 2.4 as $A_j = F_{\alpha^2}^{-1}$ with $\alpha = \sqrt{a_j}$. As this implies, this is a section with a power transmissivity of $\sqrt{a_j}$, we choose this, as in this way the effective transmissivity over the roundtrip before the sensor will be a_j. The total reflection spectrum vector can be calculated as $x = \mathcal{T}_{10}/\mathcal{T}_{00}$. A recurrent equivalent of this method, similar to the naive approach proposed at the beginning of this section, is shown in algorithm 1.

Algorithm 1 Serial Recurrent Joint spectra

Require: $r_1, \cdots, r_Q, a_1, \cdots, a_Q$
  $r \leftarrow 0, t \leftarrow 1$ ▷ N-sized vectors
  for $i \in [1, ..., Q]$ **do**
    $F \leftarrow 1/(1 - a_i r r_i)$
    $r \leftarrow r + a_i r_i t^2 F$
    $t \leftarrow \sqrt{a_i}(1 - r_i) t F$
  end for
  return r

Chapter 3

Spectrally-overlapped Fiber Bragg Gratings

A quasi-distributed sensor can be constructed by using an array of FBGs as we saw in section 2.8. The multiplexing of several FBGs can be made by WDM, which has an intrinsic limitation in the number of sensors it can support given by the source bandwidth and the bandwidth allocated to each sensor. This limitation can be ameliorated by multiplexation schemes that extend the capacity of WDM by allowing multiple FBGs to be alloted in each spectral channel. The spectral overlap of a certain number of FBGs is allowed by requiring sufficient spectral differentiation and processing the spectrum to infer the Bragg wavelength of each sensor.

The most straightforward approach is to tailor a code in the spectra that can be easily separated even in presence of overlap. To the best of our knowledge, the first proposition in this direction is ”Intensity and Wavelength Dual-Coding” [29], in which a 2-FBG system has a second dual-peak FBG tailored to have lower reflectivity and two adjacent main lobes. This allows detection of at least one of the peaks for all levels of overlap, and in consequence the unambiguous detection of each sensor spectral shift.

A similar approach but scalable to any desired number of sensor is proposed in [30], by means of orthogonal spectral codes based on binary sequences of adjacent peaks. The spectral position of each sensor can then be extracted by finding the crosscorrelation peak with its corresponding spectral code. In a follow-up work this idea is extended by adding the possibility of phase encoding each lobe [31]. Another similar technique, known as Spectrally Coded Multiplexing (SCM) [32], is based on multiplexing in the Fourier domain. In this scheme each FBG spectral shape is tailored to occupy separate channels in this new domain, whose refractive index modulation can be obtained through inverse scattering algorithms [33].

The other direction is to use standard uniform or apodized FBGs and use more computationally expensive signal processing schemes to extract the individual central wavelengths. Although this approach is generally referred simply as WDM with spectral overlap, there is no unified term to refer to this kind of demodulation scheme. Some names that reference

this same demodulation are "WDM with amplitude-wavelength dual coding" [34], "Spectral Profile Division Multiplexing" [35] and "Intensity and wavelength division multiplexing" (IWDM) [36]. There is a requirement for the spectral profiles to be sufficiently different to be discerned from each other, which is usually done through the intensity values. Although this is a good approach and most literature uses this approach of differentiation, we will use the term Spectrally-overlapped WDM (SWDM) as it includes a broader set of differentiation beyond just intensity. We will outline a more precise formulation of the problem that SWDM tries to solve, as well as survey in detail the approaches taken in the literature.

3.1 Formulation

Conventionally WDM of FBG sensors requires no overlap on the FBG spectra for conventional peak detection (CPD) techniques to have adequate results. Otherwise, there will be crosstalk between overlapped channels, and in consequence error in the peak estimation. SWDM consists in finding the spectral shifts of the sensors even when there is spectral overlap, provided that there is sufficient difference between the spectral profiles.

We can formulate SWDM in a more formal mathematical manner as follows. Given a Q-sized FBG array, with each sensor associated to an integer $i \in \mathbb{Q} \triangleq \{1, ..., Q\}$ and spectral observations of N points of the reflection spectra of the array centered in λ_0, with a span of 2Δ and a resolution of $2\Delta/N$. The objective is to obtain an estimation $\hat{y}$ of the Bragg wavelength vector $y = \{\lambda_{B_i}\}_{i \in \mathbb{Q}}$ from the spectral observation x, where the Bragg wavelength of the ith sensor is given by $\lambda_{B_i} \in \mathbb{B}\, \forall i \in \mathbb{Q}$ and $\mathbb{B} \Subset \mathbb{A}$.

To meet this objective, each sensor's reflection spectrum must be different from each others, and from all possible overlap of other sensors. This requirement is usually achieved by using different spectral peak reflectances or by using attenuators and couplers, to tailor individual effective peak reflectances for each sensor. The effective peak reflectances must jointly satisfy

$$P_i \neq P_j \wedge P_i \neq J(\{P_k\}_{k \in \mathcal{Q}_i}) \mid j \in \mathbb{Q}_i, \mathcal{Q}_i \subset \mathbb{Q}_i \, \forall i \in \mathbb{Q} \tag{3.1}$$

where P_i is the effective peak reflectance of the ith sensor, $\mathbb{Q}_i = \mathbb{Q} \setminus \{i\}$ is the set of sensor numbers with i excluded, and $J(\cdot)$ describes the joint reflectivity computation from individual reflectivities, which is dependent on the topology of the array.

We can describe the relation between the spectral shifts y and the observed spectrum x by a function $f : \mathbb{B}^Q \to [0, 1]^N$. We normally don't have direct access to the exact function but given a good knowledge of the FBG characteristics and the topology, a sufficiently good approximation $\tilde{f}(y) \approx f(y) \, \forall y \in \mathbb{B}^Q$ can be obtained, as described in section 2.9. Even with a perfect approximation, finding an analytical inverse mapping $\tilde{f}(x)^{-1} \approx y$ is not possible. In the following section, we describe the different approaches present in the literature to approximate this inverse mapping.

3.2 State of The Art

3.2.1 Naive Approach

A naive approach to solving SWDM consists in computing simulations from a grid within the range of possible values of each λ_{Bi}, and selecting the point where a metric of distance is minimized between the observed spectrum and the simulated spectrum. More formally, an estimation is obtained as $\hat{y} = \operatorname{argmin}_{\hat{y}} \mathcal{L}(\hat{x}, x) \mid \hat{x} = \tilde{f}(\hat{y})$, where $\mathcal{L}(\cdot, \cdot)$ is a distance metric that quantifies how similar the two vectors are. This approach is extremely costly but guarantees an accuracy bound given by the resolution of the grid. This method has complexity $\mathcal{O}(M) = \mathcal{O}(\mathcal{M}^Q)$, where $\mathcal{M}$ is the sampling resolution over each λ_{Bi}.

Following this naive approach, minimum variance shift (MVS) [22] has been proposed to solve the problem for two overlapped FBGs. This method uses as a distance metric the variance shift, defined as the integral of the squared difference. The authors improve over the naive approach by performing a coarse sampling followed by a fine sampling around the suboptimal solution. On a follow-up work [34], the authors improve MVS by modifying the refinement step, which consists of an initial refinement of the more salient FBG[12] and then a refinement of the other FBG, this is followed by joint refinement of both FBGs over a finer grid with a narrower range.

In this work, we implement a regressor called `lookuptable` (LUT) which follows this naive approach. It is essentially a key-value database where the query, value & keys correspond to x, $\hat{y}$ & $\hat{x}$. This means that, we have a database $\{\hat{X}, \hat{Y}\}$ composed from $\{\hat{x}_m, \hat{y}_m\}$ pairs, where $\hat{x}_m = \tilde{f}(\hat{y}_m)$. For an observed spectrum x, we retrieve $\hat{y}_k \mid \mathcal{L}(x, \hat{x}_k) \leq \mathcal{L}(x, \hat{x}_m)\ \forall m \in \{1, \ldots, M\}$. As we stated before, this approach has a significant time complexity if the distances are computed sequentially. Instead, we precompute the spectra and calculate the distances in a batched manner. In this way, we leverage the parallelization capacity and translate the time complexity into memory complexity.

3.2.2 Evolutionary Algorithms

One approach to retrieve the peak location of overlapping FBGs is through evolutionary algorithms (EA) [37]. Provided that one has a good approximation of the discrete spectrum $x \approx \tilde{f}(y)$, the objective is to find $\hat{y}$ that yields an approximated discrete spectrum $\hat{x} = \tilde{f}(\hat{y})$ similar to the observed discrete spectrum x, generally quantified by the ℓ_1 or ℓ_2 norm. The optimization problem can be formulated as

$$\hat{y} = \operatorname*{argmin}_{\hat{y}} \mathcal{L}(x, \hat{x}) \tag{3.2}$$

[12] Here the comparison is between the observed spectrum and the simulated single FBG spectra

where $\mathcal{L}(\cdot,\cdot)$ is a loss function that quantifies the dissimilarity between its inputs. This optimization has to be solved by sampling $\hat{y}$ until an adequate solution is found. Evolutionary algorithms solve this problem with faster convergence than random or grid search by leveraging ideas found in biological evolution. These algorithms have great performance but require a significant processing time since there is no training process involved and the minimization problem has to be solved for each inference. The basic approach is the genetic algorithm (GA). Proposed in this context by Shi *et al.* [38]. This is an iterative population-based optimization algorithm based on the crossover of genes, random mutations, and selection of best individuals for the next generation until a stop policy is reached. Shi *et al.* use binary genes and make no assumption about the effective peak reflectance value, making the genes contain λ_{B_i} and also P_i information.

In follow-up work, Shi *et al.* propose the use of Simulated Annealing (SA) [39]. SA is an iterative optimization algorithm based on the probabilistic improvement of a single candidate with binary genes by random mutation. The candidate is replaced with probability proportional to the improvement and inversely proportional to a temperature parameter. The temperature parameter is decreased along the procedure to increase the probability of acceptance. This approach is described by algorithm 2. In contrast to GA, here the genes code only for λ_{B_i} as do all other algorithms.

Algorithm 2 Simulated Annealing Algorithm

Require: I, J, T_0
  $T \leftarrow T_0, s \leftarrow sample()$
  for $i = [1, \ldots, I]$ **do**
    for $j = [1, \ldots, J]$ **do**
      $s_{new} \leftarrow mutate(s)$
      $\Delta E \leftarrow g(s_{new}) - g(s)$
      if $z \sim Bernoulli(sigmoid(\Delta E/T))$ **then**
        $s \leftarrow s_{new}$
      end if
    end for
    $T \leftarrow T\eta$
  end for
  $y \leftarrow binary\_to\_decimal(s)$

Dynamic Multi-Swarm Particle Swarm Optimization (DMS-PSO) [40] has also been proposed in the literature. DMS-PSO is an extension of Particle Swarm Optimization (PSO), being an EA based on a swarm of particles (population) whose position is updated by a velocity with inertia and two attractors to the best previous position of the particle and the best position of the population so far. The velocity for the jth particle of the ith population is given by $v_j^i = \omega v_j^i + c_1 r_j^i (pb_j^i - y_j^i) + c_2 q_j^i (gb^i - y_j^i)$, where ω is a weight constant, c_1 and c_2 are acceleration constants, r_j^i and q_j^i are two random values drawn from a uniform distribution in $[0, 1]$, pb_j^i is the best y of the jth particle of the ith population, and gb^i is the best y of the ith population. In DMS-PSO, the swarm is divided into sub-swarms that evolve independently and are regrouped after a certain amount of iterations.

In follow-up work, Tree Search Dynamic Multiswarm Particle Swarm Optimization (TS-DMS-PSO) [41] is proposed to improve the convergence time of DMS-PSO by dividing the problem into M phases, starting with a reduced number of spectral points L_1 and a full solution space $\mathcal{S}_1 = \mathbb{B}^Q$. In each phase, the search space of the previous phase $\mathcal{S}_{m-1} = S_{(m-1)1} \times \ldots \times S_{(m-1)Q}$ is updated as $\mathcal{S}_m \Subset \mathcal{S}_{m-1}$, and the number of spectral points is increased to $L_m > L_{m-1}$. The space is updated through the rule

$$S_{mi} = \left[max\left(S^{min}_{(m-1)q}, y^{bsf}_q - \frac{S^{diff}_{(m-1)q}}{10} \right), min\left(S^{max}_{(m-1)q}, y^{bsf}_q + \frac{S^{diff}_{(m-1)q}}{10} \right) \right] \tag{3.3}$$

where y^{bsf}_q is the qth dimension of the best candidate so far and $S^{diff}_{mq} \triangleq S^{max}_{mq} - S^{min}_{mq}$.

Self-adaptive Neighborhood search differential evolution (SaNSDE) is proposed in [42]. It builds over the differential evolution (DE) algorithm, an EA that modifies the GA mutation step to generate a differential vector as $V_i = Y_a + F(Y_b - Y_c)$, where a, b and c are chosen randomly, given that $a \neq b \neq c \neq i$, with Y_j being the gene of the jth individual and $F \in [0, 2]$ being a constant factor. Then, in the crossover step a trial vector U_i is generated by randomly replacing elements of Y_i by V_i's with probability given by $CR \in [0, 1]$, if the fitness of U_i is better than Y_i's it replaces the former.

SaNSDE is an attempt to merge the ideas of two modifications of the canonical DE algorithm, the first is Self-adaptive Differential Evolution (SaDE) [43], which proposes to modify the mutation step to choose randomly between two options as

$$V_i = \begin{cases} Y_a + F(Y_b - Y_c) & \text{if } x \sim Uniform(0, 1) < p \\ Y_i + F(Y_{best} - Y_i) + F(Y_a - Y_b) & \text{otherwise} \end{cases} \tag{3.4}$$

and having non fixed parameters F, CR which are drawn from normal distributions. The mean of the crossover rate distribution μ_{CR} and p are updated every K iterations based on the performance of the individuals. The second incorporated approach is Neighborhood Search Differential Evolution (NSDE) [43], which proposes to draw F from a normal distribution with probability fp otherwise from a Cauchy distribution, following in this way an explore or exploit strategy. These strategies are merged in SaNSDE by modifying SaDE to draw F, as it is done in NSDE and update fp in the same manner p is updated [43].

The Swap Differential Evolution (SDE) algorithm is proposed in [24] as an extension of DE. The performance is improved by leveraging the symmetry of the search space of the problem, by noticing that for the optimal solution there are $N! - 1$ suboptimal solutions that correspond to the permutations of the solution vector. The authors solve this by adding a swap step after the crossover step, where two random elements of the trial vector are swapped and if the fitness is improved it replaces the former.

Another DMS-PSO alternative is proposed in [44], in which it is slightly modified to

exploit the symmetry of the local and global optima by searching the space of all permutations at the end of the algorithm. In another work, particle swarm optimization based simulated annealing (PSO-SA) [44] is proposed, in which ideas of SA are applied to PSO. A judgment step is added where the particle position is updated if an acceptance criterion is passed, as done in SA. A temperature parameter is decreased in every epoch in order to relax the acceptance criterion.

The Distributed Estimation Algorithm (DEA) [23] is an EA where a population is sampled from a Mixture of Gaussians (MoG) probability distribution. The distribution is uniform in the first step and is then modified by updating the MoG with the $m < n$ top fitness individuals of the population, where n is the size of the population. The algorithm is iterated until the termination condition is reached.

3.2.3 Regression

The other approach to solve SWDM is to build a regression model [45]. In this approach, the intent is to invert the function $x = f(y)$ that generates the observed spectra. Since it is not possible to find an analytical solution to the problem an approximation is used instead. The approximation is given by a parameterized function $g_\theta(x) \approx y = f^{-1}(x)$, characterized by the parameter vector θ. The optimal vector θ is found by minimizing a loss function over an M-sized paired dataset $Y = \{y_i\}_{i=\{1,\dots,M\}}, X = \{x_i\}_{i=\{1,\dots,M\}}$, where $x_i = f(y_i)$. The loss function quantifies the difference between the predictions and the actual values, usually through the ℓ_1 or ℓ_2 norm. This is formalized as

$$\theta = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(Y, g_\theta(X)) \tag{3.5}$$

This optimization problem can be solved either through gradient descent or by finding an analytical solution under certain formulations. This approach has, in general, lower accuracy compared to EA but has an inference time that is significantly shorter since there is a previous learning step.

The use of a Self-Organizing Feature Map (SOFM) is proposed in [46]. It consists in an unsupervised clustering algorithm where each output of a network is associated with a cluster in the input space. Each output is calculated by taking a distance metric of its weight and the input. The weights are learned by finding the minimum output for an input and updating its weight to be close to it, as well as its neighboring nodes. This generates a semantic mapping where similar inputs are mapped closer together and dissimilar ones are far apart. In this context, SOFM is used as a LUT similar to the one proposed in section 3.2.1 but the clusters are used to form the query-value pairs instead of directly using a grid.

The use of Extreme learning machine (ELM) to obtain the spectral position of a series of FBGs is reported in [47]. This regression model is a single layer perceptron followed by a weighted sum. This can be represented as $g(x) = \sigma(Wx + b)\beta$ where W and b account for a fixed random affine transform, $\sigma(\cdot)$ is the sigmoid function and β is a ponderation vector.

A system of equations from the dataset as $H\beta = Y$ is constructed, where $H = \sigma(WX + b)$. From this, an estimation of β is made by using the Moore-Penrose generalized inverse as $\hat{\beta} = H^{\dagger}Y$. The use of ELM in this context is also reported in [48].

In [49] the use of Least Squared Support Vector Regression (LS-SVR) is reported. LS-SVR is a regression model based on a linear transformation of data projected to a high-dimensional space, thus achieving a nonlinear regression in the low-dimensional space. This is formalized as $\hat{y} = g(x) = w^T\phi(x) + b$, where w^T and b account for an affine transform and $\phi(x)$ is a function that takes x to a higher dimensional space. The optimization problem is formulated as

$$\min_{w,b,e_i} \frac{1}{2}||w||^2 + \frac{\gamma}{2}\sum^{N} e_i^2 \qquad (3.6)$$
$$s.t \quad y_i = g(x_i) + e_i$$

This can be solved in dual space and with no need to take the data to a higher dimension by use of a kernel $k(x_i, x_j) = \phi(x_i)\phi(x_j)$. This is known as the "kernel trick".

More recently, Deep Learning solutions have been proposed. For details on the underlying principles and functioning of these algorithms refer to section 4. The use of standard NNs is reported in [50]. The use of RNNs is reported in [51], [52], where LSTM are used, and in [53] the use of GRU is proposed. The use of a CNN is proposed in [54] and the use of Dilated CNN is proposed in [55]. In [56] an AE CNN is proposed to reduce the required dataset size, by pretraining on unlabeled data and then training a neural network on top of the AE Encoder. The accuracy is further improved by refining the output through the use of DE, which converges rapidly since a good coarse solution is given.

In a SWDM FBG array, it is generally desirable to achieve subpicometer resolution in order to be on par with CPD resolution. On the other hand, the processing time must be in the order of the sampling period of the spectrogram, as otherwise the improvement in the number of sensors would not be justified in comparison with the use of a time-wavelength division multiplexing scheme. It is difficult to put the results of the proposed methods in the literature into fair perspective, as the results are heavily dependent on the experimental setup and the used processing hardware and not all the authors report the same metrics. Regardless of this, there is a clear difference between the EA and the regression models. Most EA models achieve subpicometer resolution consistently but with great processing time in the order of seconds. On the other hand, regression models achieve resolutions in the order of picometers but with a clear dependence of the error over the overlap of sensors although with a consistently lower processing time in the order of milliseconds even for inefficient CPU implementations. A table with all the reviewed methods is shown in table 3.1.

Table 3.1: SWDM Algorithms review. Alg., short for Algorithm, displays the acronyms of the algorithms. Sim., short for Simulation, has the FBG approximation used for simulation. Diff., short for Difference, specifies the used spectral differentiation between sensors in the array. Min., short for Minimization, signals the minimization objective of the algorithm. The minimization can be Squared Reconstruction Error (SRE), Absolute Reconstruction Error (ARE) or Mean squared Error (MSE); the first two refer to differences between the observed and reconstructed spectra, while the latter represents the error between the real and inferred spectral positions. Exp., short for Experimental, informs if experimental data was used.

Year	Method	Authors	Ref.	Topology	Alg.	Q	Sim.	Diff.	Min.	Exp.
2002	Search	Gong *et al.*	[22]	Parallel	MVS	2	Gaussian	none	SRE	Yes
2003	Search	Chan *et al.*	[34]	Parallel	MVS	2	Gaussian	Intensity	SRE	Yes
2005	Regression	Zeng *et al.*	[46]	Parallel	SOFM	2	Gaussian	Intensity	SRE	Yes
2003	Evolutionary	Shi *et al.*	[38]	Parallel	GA	2	Gaussian	Intensity	SRE	Yes
2004	Evolutionary	Shi *et al.*	[39]	Parallel	SA	2	Gaussian	Intensity	SRE	Yes
2005	Evolutionary	Liang *et al.*	[40]	Parallel	DMS-PSO	2 to 10	Gaussian	Intensity	SRE	No
2006	Evolutionary	Liang *et al.*	[41]	Parallel	TS DMS-PSO	2 to 100	Gaussian	Intensity	SRE	No
2011	Evolutionary	Liu *et al.*	[42]	Parallel	SaNSDE+	2	Not Clear	Intensity	ARE	Yes
2013	Evolutionary	Jiang *et al.*	[24]	Serial	Swap DE	2	Uniform	Intensity	ARE	Yes
2014	Regression	Jiang *et al.*	[47]	Parallel	ELM	2	Gaussian	Intensity	MSE	Yes
2014	Regression	Chen *et al.*	[49]	Parallel	LS-SVR	2 to 30	Gaussian	Intensity	MSE	Yes
2017	Evolutionary	Guo *et al.*	[35]	Serial	DMS-PSO	2 to 4	Uniform	Intensity	ARE	Yes
2018	Regression	Manie *et al.*	[48]	Parallel	ELM	3	Gaussian	Intensity	MSE	Yes
2018	Evolutionary	Qi *et al.*	[44]	Parallel	PSO-SA	2 & 4	Gaussian	Intensity	SRE	Yes
2019	Evolutionary	Zhou *et al.*	[23]	Parallel	DEA	2 to 10	Uniform	Intensity	ARE	Yes
2019	Regression	Jiang *et al.*	[51]	Parallel	LSTM	2	Gaussian	Intensity	MSE	Yes
2020	Regression	Li *et al.*	[54]	Parallel	CNN	2	Gaussian	Intensity & BW	MSE	No
2020	Regression	Manie *et al.*	[53]	Parallel	GRU	4	Gaussian	Intensity	MSE	Yes
2020	Regression	Manie *et al.*	[52]	Parallel	LSTM+denoise	4	Gaussian	Intensity	MSE	Yes
2020	Regression	Wang *et al.*	[50]	Parallel	NN	2	Gaussian	Intensity	MSE	Yes
2021	Regression	Li *et al.*	[55]	Parallel	Dilated CNN	2 & 4 & 8	Gaussian	Intensity & BW	MSE	No
2021	Regression	Chiu *et al.*	[56]	Parallel	AE CNN + DE	3 & 5 & 7	Gaussian	Intensity	MSE	Yes

Chapter 4

Artificial Neural Networks

Neurons were first proposed as the functional ”atom” of the nervous system by Santiago Ramón y Cajal in the XIX century [57], under the ”Neuron Doctrine” theory. This led to the connectionism approach of cognitive neuroscience, which proposes that mental phenomena are a consequence of the interaction of an interconnected network of simple units, that perform basic computations. This approach proved to be a powerful paradigm to construct Artificial Intelligence (AI) systems through the advent of Artificial Neural Networks (ANNs).

Neurons are specialized cells capable of integrating and relaying information from other neurons. A depiction of a neuron is shown in figure 4.1. They are the functional substrate of neural circuits in almost all animals. Each neuron is capable of basic computation by integrating inputs from other neurons, which trigger the transmission of voltage spikes through their axons. This, in turn, transmits information to other neurons through their synapses. Chemical synapses are the predominant type of synapse, where special molecules, called neurotransmitters, are excreted and captured by dendrites of other neurons. Neural circuits are capable of achieving complex tasks such as perception, memory, motor coordination and planning.

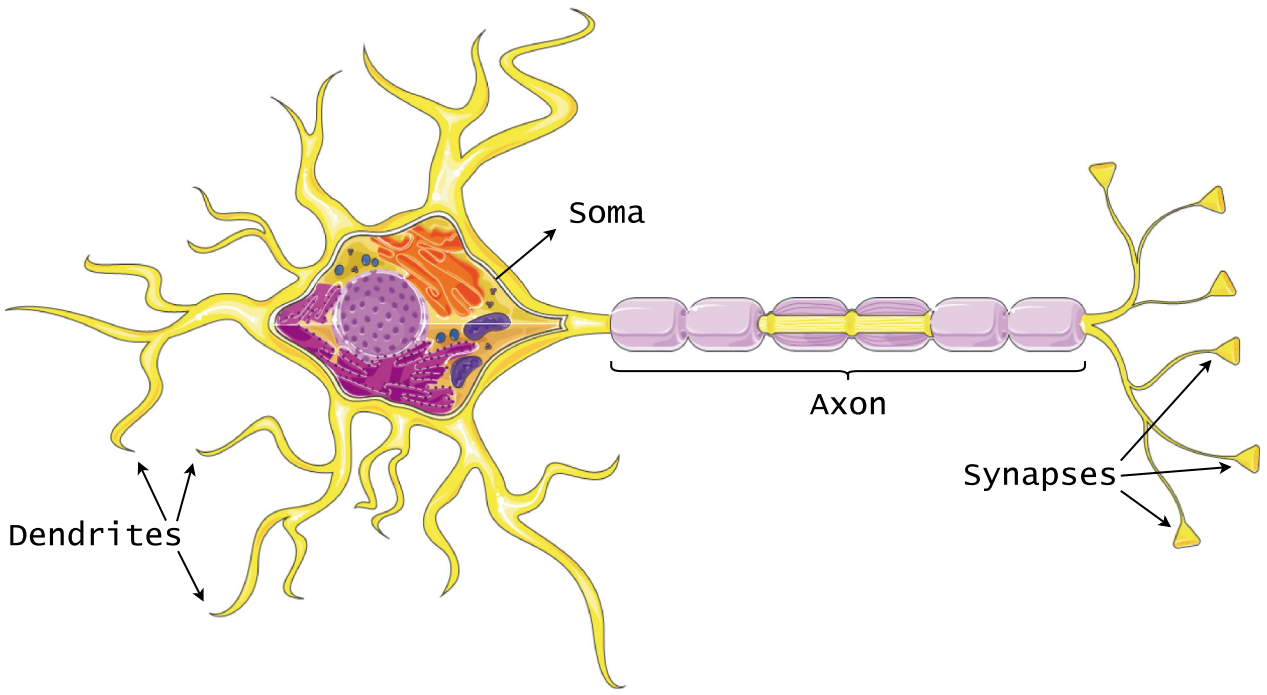


Figure 4.1: Myelinated multipolar neuron illustration.

ANNs are a particular type of parameterized Machine Learning (ML) models which have gained a lot of traction in the last decade. Although proposed in the previous century, they could not meet the expected results, making researchers lose interest in this approach. We are now in a renaissance of this paradigm thanks mainly to two key factors, *(i)* the increase in parallel compute capacity through the advances in GPU technology and *(ii)* the availability of high volume and high-quality data in multiple domains. These factors have allowed faster training and bigger architectures previously unfeasible. In this new wave of interest, this technology has been re-branded as Deep Learning (DL), alluding to the increasing depth of the models and to the data-driven optimization procedure by which they are constructed.

The universal approximator theorem [58] states loosely that an arbitrary function can be approximated to any desired level of accuracy with a sufficiently large model. This is the basis for using ANNs for simple or difficult problems. This theorem roughly states that there is an adequate architecture that can represent a function, but it does not give a specific size or depth for a particular problem nor certainties over the optimization of its parameters. Finding an adequate ANN model for a given problem is not straightforward and relies heavily on heuristics and current trends.

In this chapter, we intend to give a comprehensive overview of DL and the concepts required to understand the methods underlying the proposal of this Thesis. For more in-depth reviews of DL refer to [58], [59].

4.1 Machine Learning

Machine Learning (ML) refers to a subfield of AI, which relays on learning as a means of understanding data for a given task. Learning can be Supervised, where a mapping present in some phenomena $f : \mathcal{X} \rightarrow \mathcal{Y}$ is approximated by a function $\hat{f} \in \mathcal{F}$. The sets $\mathcal{X}$ and $\mathcal{Y}$ correspond to the input and output domains respectively, and $\mathcal{F}$ is the hypothesis set. The

optimal function is found in a data-driven manner by minimizing a distance metric $\mathcal{L}(f, \hat{f})$. This is carried out through an optimization procedure or by finding an analytical solution that solves the optimization problem. Some examples are Linear Regression, Support Vector Machines (SVM) and Decision Trees [60]. Alternatively, there is also Unsupervised learning in which a data distribution is structured in a way that can be useful for other task or for human understanding. Some examples of this are clustering algorithms, such as K-means or Mixture of Gaussians (MoG), or dimensionality reductions algorithms, such as Principle Component Analysis (PCA), Uniform Manifold Approximation and Projection (UMAP) and T-distributed Stochastic Neighbor Embedding (T-SNE). For a more formal description of these concepts refer to [61].

The kind of problems solved by supervised machine learning algorithms can be separated either in regression or classification problems. Regression problems describe finding a mapping $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are compact continuous sets. In general, they are assumed to be $\mathcal{X} \in \mathbb{R}^N$ and $\mathcal{Y} \in \mathbb{R}^M$, where N and M are the input feature size and target size, respectively.

On the other hand, a classification problem describes finding a mapping $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y}$ is a discrete set of size M, and we wish to classify an instance $x \in \mathcal{X}$ into its corresponding class $y \in \mathcal{Y}$. The usual approach taken in ML to solve this problem consists in assigning an order for the classes and applying a binary mapping over the target set. The mapping $O : \mathcal{Y} \to \mathcal{Y}^*$ takes an instance $y = y_m$, where y_m corresponds to the mth class, and represents it as y^*, a binary vector of length M with the mth dimension set to one and the rest to zero. In this representation, the mth dimension represents if the instance belongs to the mth class. This is referred to as *one-hot-encoding*. The concept of encoding will be described in more detail in section 4.10.4.

As can be seen, classification is turned into regression by encoding the target. Because of this equivalence and for simplicity, we will generally describe ANNs as regressors, and will refer to the classification setting explicitly just as a change in perspective. This will be particularly useful when describing the kinds of space transformations ANNs perform over the input space.

4.2 Perceptron

The perceptron is the overly simplified equivalent of a neuron. It relays on the assumption that *(i)* the frequency of spike firing is the only relevant signal, *(ii)* the connection is unidirectional from input to output, *(iii)* the input and output are deterministic and *(iv)* the output is binary. The perceptron is then essentially just a binary classifier.

A perceptron is constructed from an affine transformation followed by a nonlinear function. This can be represented as

$$y = f(Wx + b) \tag{4.1}$$

where x and y are the input and output vector of lengths N and M respectively, W is the weight matrix of size $M \times N$, b is the bias vector of length M, and $f(\cdot)$ is an activation function, in particular a sign function that outputs 0 if below zero or 1 otherwise. This is illustrated in figure 4.2, where the computation of a single output is highlighted, analogous to a single neuron in its biological counterpart. The mth output is given by

$$y_m = f\left(\sum_{n=1}^{N} x_n w_{nm} + b_m\right) \tag{4.2}$$

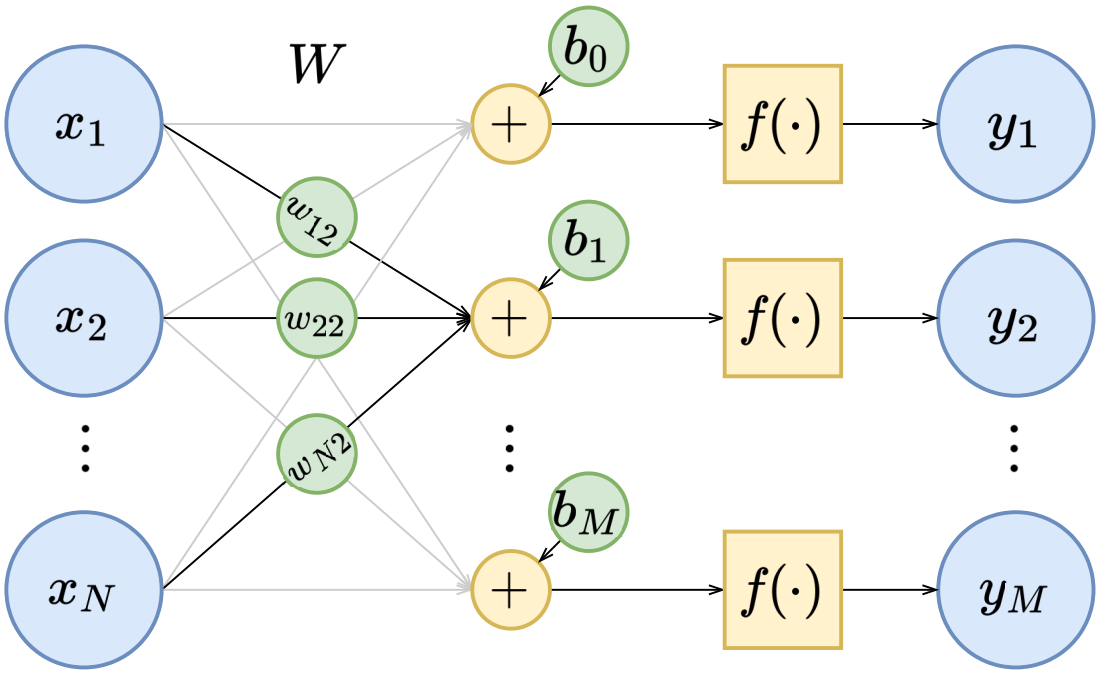


Figure 4.2: A Perceptron with an N-sized input and M-sized output. For clarity, only the weights between the input and the 2th output are depicted.

4.3 Multilayer Perceptron

A single perceptron is not a good function approximator by itself. Instead, the stacking of multiple layers enable the increase in approximation capacity to arbitrary requirements. The essential configuration is the Multilayer Perceptron (MLP), also referred in the context of NNs as Feedforward Neural Networks, Fully Connected Neural Networks (FC NNs) or Dense Layers. As the name suggests, it is just a number of perceptrons stacked on top of each other, with the output of a layer being the input of the following one. Furthermore, in an MLP other nonlinear activation functions are used instead of the sign function. These will be explained in more detail in section 4.5. The output of the ith layer is given by

$$z_i = f(W_i z_{i-1} + b_i) \tag{4.3}$$

where W_i and b_i are the weight and bias of the ith layer. The input is $x = z_0$ and the output is $y = z_{K+1}$, where K is the number of hidden layers. An arbitrary MLP with K hidden layers can be illustrated by figure 4.3

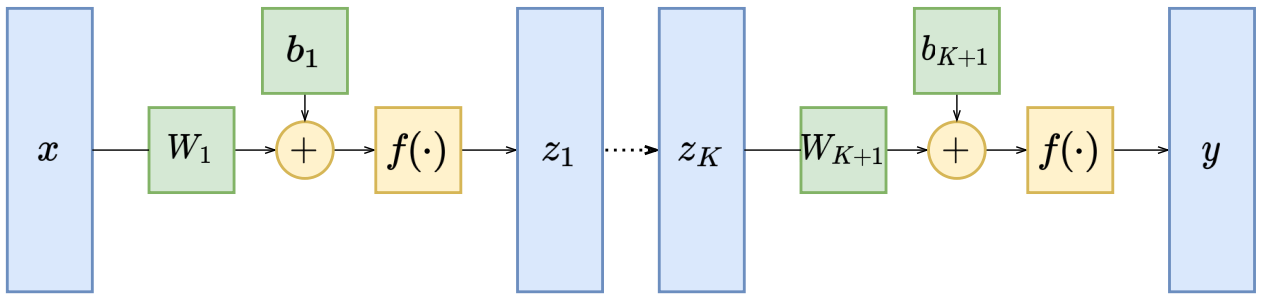


Figure 4.3: A $K+1$ layer MLP with K hidden layers.

In the context of regression, an MLP can be understood as a soft piece-wise approximator, which leverages the linear part of the nonlinearity for each of the pieces [61]. In the context of classification, the affine transformation of a perceptron represents essentially a space transformation bounded only to uniform contraction or expansion, rotation and translation. If there was no nonlinearity, having multiple layers would have the same representation power of a single layer, as multiple consecutive affine transformations can be expressed by a single equivalent affine transformation. The compound transformation of the MLP is a nonlinear transformation that can make an input distribution linearly separable over the set of classes of the classification task.

The classification perspective is clearly illustrated for a simple example in figure 4.4. In this example, a two-dimensional input is classified into two classes, blue or red. The blue and red curves display the distributions of each class, i.e. the possible values they can take. The hidden representation is 2-dimensional and the grid displays how the space is morphed. The input distributions are morphed in the hidden layer and can be separated by a line, i.e. the distributions are linearly separable. Then the inference of the class can be made by an affine transformation over the hidden representation followed by a threshold, represented as the blue and red regions in the hidden space.

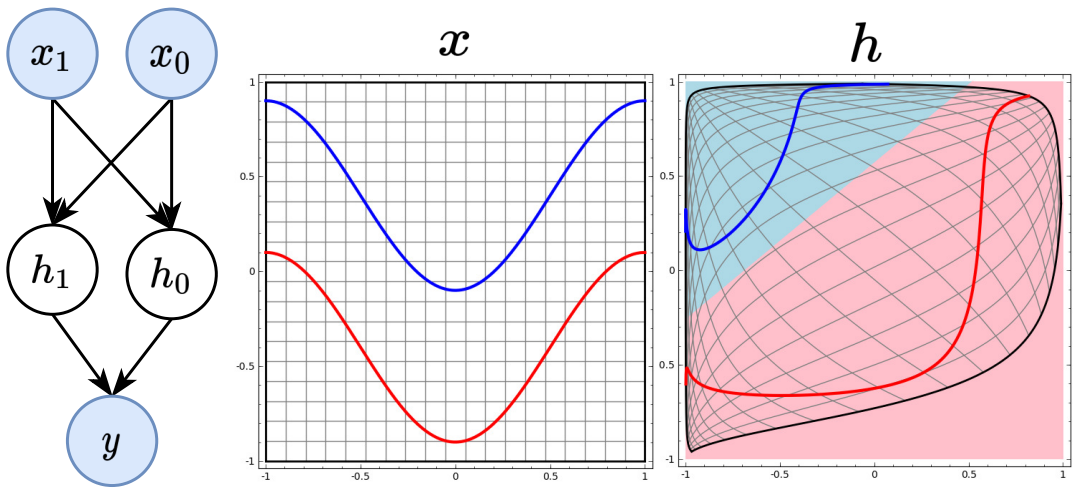


Figure 4.4: Single layer MLP with two inputs, one hidden layer of size 2 with sigmoid activation function and a single binary classification output. It is illustrated how a non-separable class distribution can be nonlinearly transformed to be linearly separable. Taken from [62] which explains in greater detail this perspective.

4.4 Backpropagation

The learning procedure used in DL is generally Backpropagation. It consists in updating the weights of the model through gradient descent. Gradient descent is an optimization procedure in which a critical point of a function, ideally a global minima, is achieved by choosing a random initial point in the domain and updating the position by taking a step in the opposite direction of the derivative, weighted by a parameter referred to as learning rate. This update is repeated in an iterative manner to arrive at a critical point where the derivative decreases. The optimization of a simple scalar function is shown in figure 4.5.

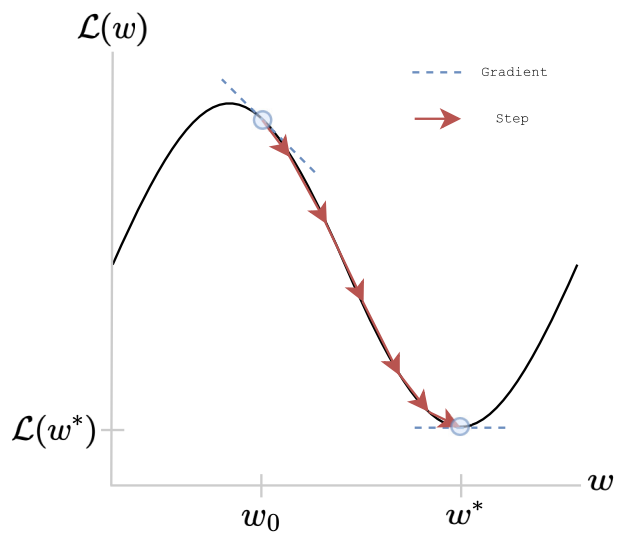


Figure 4.5: Simple gradient descent example. Here w_0 is the initial point that is updated by taking a series of steps to reach the minimum w^*

This is more formally stated in the following way. Given a differentiable function $J : \mathcal{W} \to \mathbb{R}$, gradient descent is an iterative optimization process that finds a critical point $w^* \in \mathcal{W} | \nabla_w J(w^*) = 0$. It consists in choosing an initial random point $w_0 \in \mathcal{W}$ and following the iterative process described by

$$w_i = w_{i-1} - \eta \nabla_w J(w_{i-1}) \tag{4.4}$$

where w_j is the point obtained in the jth iteration and η is the learning rate parameter. This process is iterated for a given amount of iterations or until improvement stagnates. For the case of a NN, w represents its parameters and $\nabla_w J(w)$ is the gradient of the loss function with respect to the parameters.

The path a solution takes during gradient descent is smoothly displayed for an example loss landscape in figure 4.6. In contrast to the previous example from figure 4.5, here the loss landscape is ”complex” as it is not strictly convex, and has saddle points and multiple local minima. This is generally the case in DL problems.

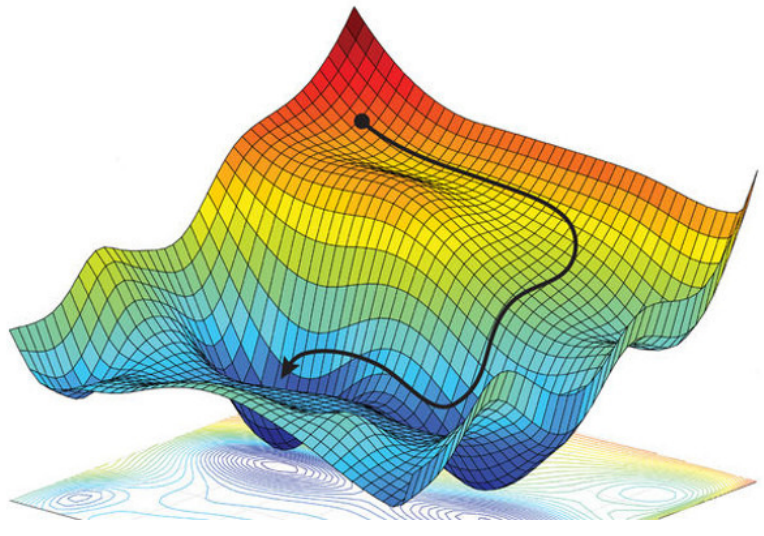


Figure 4.6: Gradient descent in a complex loss topology.

The gradients are calculated through the chain rule by calculating the partial derivatives from output to input. This is termed ”backward pass”. Backpropagation stands for backward propagation of errors and describes this procedure for calculating the gradient and updating parameters through gradient descent.

The use of the chain rule in a FC NN with nonlinearity given by $f(\cdot)$ is illustrated for a simple example in figure 4.7 and a more complex example in figure 4.8. The partial derivatives in green give the expressions for the derivatives of each weight. The downstream arrows represent what is termed ”forward pass” and corresponds to calculating each layer output from top to bottom. The upstream arrows represent the backward pass that corresponds to the partial derivative calculation, which is performed from bottom to top. As can be seen from these two examples, it is necessary to calculate all the previous gradients to calculate the gradient of one layer, hence the name backpropagation, as the error given by the output loss is propagated from bottom to top.

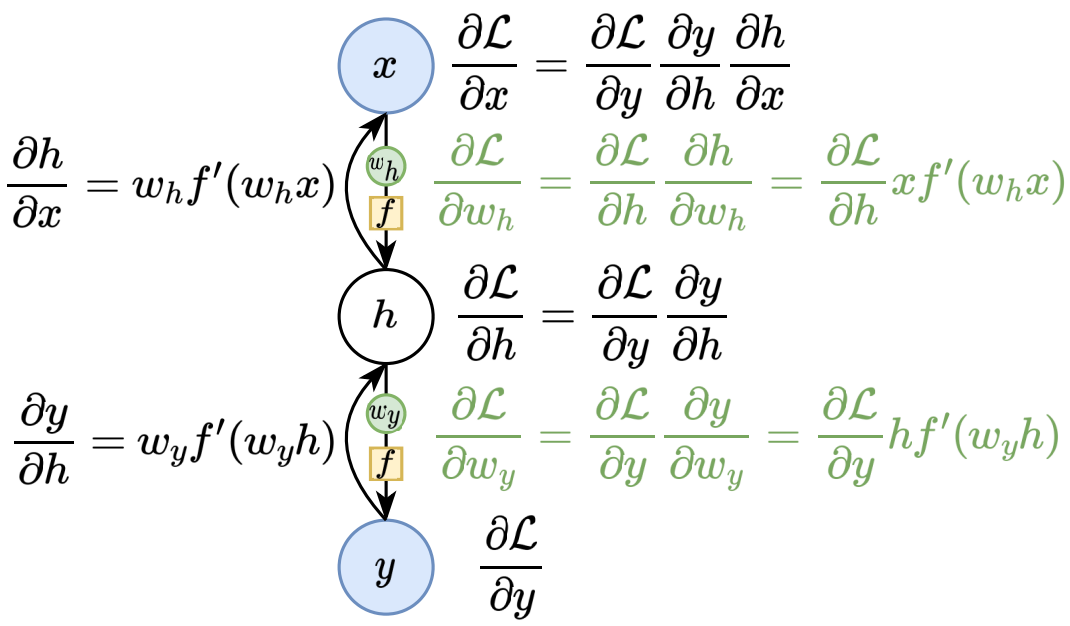


Figure 4.7: Chain rule applied to a simple FC NN.

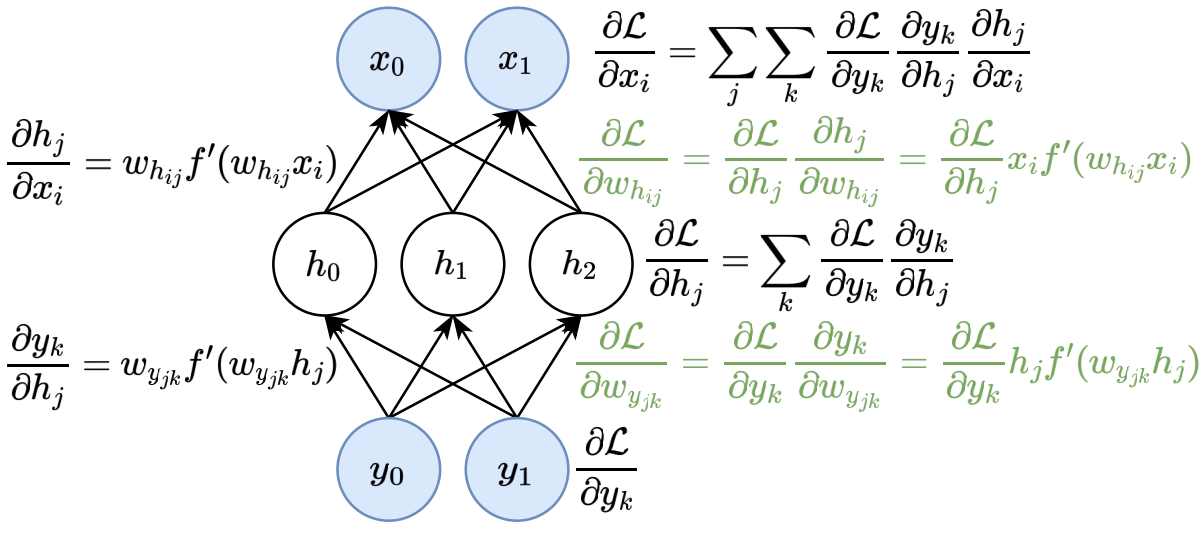


Figure 4.8: Chain rule applied to a more complex FC NN. $w_{h_{ij}}$ represents the weight between input x_i and hidden node h_j and $w_{y_{jk}}$ represents the weight between hidden node h_j and output y_k. The arrows and weights of the forward pass are obviated for clarity but are equivalent to the simple NN example.

The loss function in NNs is a measure of performance for a given task or a surrogate of it (in case it is not suitable for gradient methods). Given a dataset $\{X, Y\}$ comprised of a set of input target pairs $\{x_i, y_i\}$, the loss over a single pair is given by $\mathcal{L}(f_w(x_i), y_i)$ and the optimal parameters are given by

$$w = \underset{w}{\operatorname{argmin}} \mathcal{L}(f_w(X), Y) \tag{4.5}$$

where $\mathcal{L}$ is a loss metric adequate for the task. Usually Mean Squared Error (MSE) or Mean Absolute Error (MAE) are used for regression problems and the Binary Cross Entropy (BCE) is used for classification problems.

In practice, Stochastic Gradient Descent (SGD) is used, where in each epoch the dataset is divided into batches, and in each iteration the loss over a batch is calculated and the parameters are updated accordingly. This is done for two reasons. One is a computation constraint due to limited memory resources, making it often infeasible to use the full dataset. On the other hand, SGD introduces stochasticity, which in practice allows for improved convergence. In general, the loss landscape is highly non-convex with saddle points and local minima. Furthermore, there are no general guarantees of convergence to the global optima or over the rate of convergence. Despite this, it has empirically shown in practice to achieve excellent results. Some theoretical explanations for this can be found in [61]. In the context of NNs, SGD is referred to as an optimizer. Improvements over standard SGD are often used, which are referred to as adaptive optimizers and will be explained in detail in section 4.9.1.

4.5 Activation Functions

As we stated in the previous section, MLPs with no nonlinearities can only represent linear functions and consequently, a suitable nonlinearity is required to approximate a nonlinear function. In this section, we will outline the types of activation functions and their characteristics together with some examples, in order to have a general understanding of why a particular activation function is chosen.

The identity activation function is equivalent to no activation function. It is useful for the output layer of regression as it allows to have unbounded outputs, i.e. $y \in \mathbb{R}^N$.
The step function [61] is the function used in the vanilla perceptron, which is just a heaviside function centered at the origin. It is inspired in its biological counterpart by representing its two states, firing or not firing. In practice, it has a limited representation capacity and is not suitable for gradient-based optimization, due to its constant null gradient and discontinuity at zero.

The sigmoid function can be thought of as a continuous approximation of the step function. It generally refers to any *S*-type function that is close to a step function. More formally, it refers to any function that is monotonic, bounded and has only one inflection point. In the context of DL it refers unequivocally to the logistic function [61] defined as

$$\sigma(x) = \frac{1}{1 - e^{-x}} \tag{4.6}$$

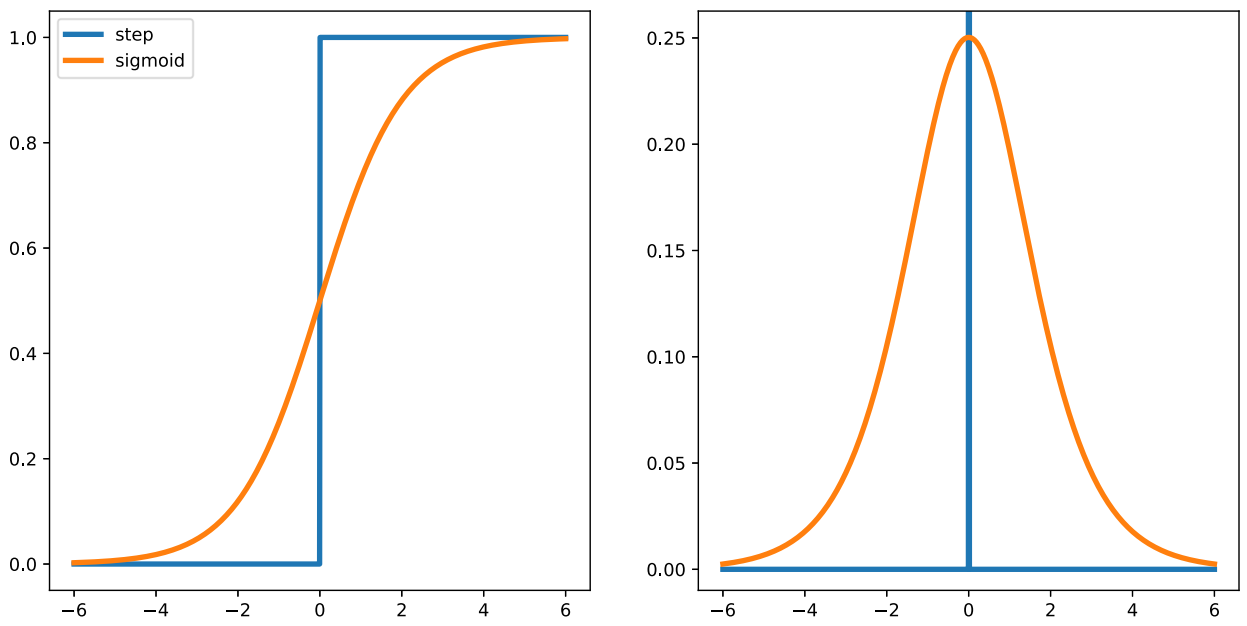


Figure 4.9: Sigmoid and step functions (left) and their corresponding gradients (right).

This function maps real line $\mathbb{R}$ into the range $[0, 1]$ with asymptotes of zero to the left and one to the right, as illustrated in figure 4.9. This function has the advantage over the step function of being differentiable. This makes it suitable for gradient-based methods, and because of this, it was a usual choice early on. One of the issues of the sigmoid function is the problem of vanishing gradients, which arises as a consequence of the decreasing derivatives as the function saturates either to one or to zero, which results in the stagnation

of gradient-based learning approaches. A similar activation function to the sigmoid is the hyperbolic tangent [61], which maps the real line into the range $[-1, 1]$ and is described by

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.7}$$

A different set of functions are the hokey-stick functions, which, as the name suggests, have a shape similar to a hockey stick. In general, they are characterized by having an unbounded monotonical part for positive inputs, usually asymptotically linear, and saturation to a constant to the left. The main exponent of this family is the Rectified Linear Unit (ReLU) [61] characterized by:

$$f(x) = \begin{cases} x & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

This function maps the real line into $\mathbb{R}_0^+$ by setting negative values to zero, as illustrated in figure 4.10. This solves the vanishing gradients problem of the sigmoid function as the derivative is constant and not zero for all positive values. Another advantage of this function is its computational simplicity, making it suitable for constructing very large models. One issue this function introduces is the ”dying ReLU” that arises from its null derivative for negative values. This characteristic can be good since it promotes sparsity of activations, i.e. many values will be exactly zero, but a pathological scenario can happen when units output zero for all inputs and consequently are unable to be updated.

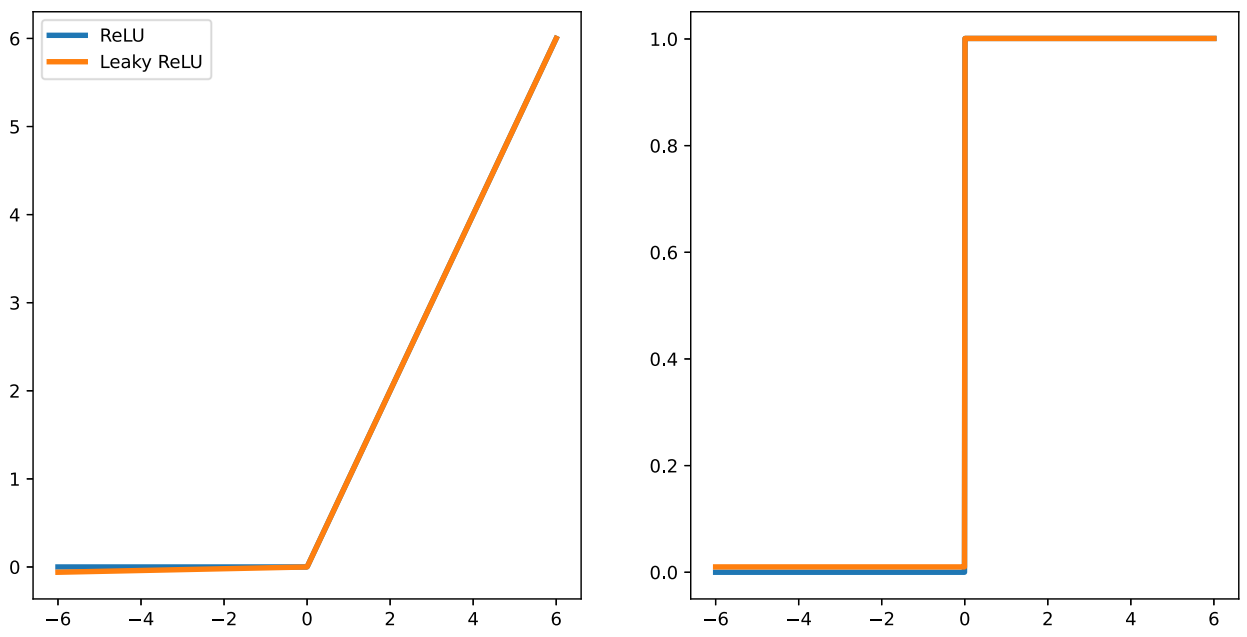


Figure 4.10: ReLU and Leaky ReLU functions (left) and their corresponding gradients (right). The value $\alpha = 0.01$ is used for the leaky ReLU.

One alternative to solve the dying ReLU issue is the Leaky ReLU [63], which is a ReLU that for negative values has a derivative given by a hyperparameter $\alpha \in [0, 1]$ and is given by the expression

$$f(x) = \begin{cases} x & \text{if} \quad x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{4.9}$$

An adequate value for α must be found, although a default value of $\alpha = 0.01$ is usually used. The leaky ReLU is illustrated in figure 4.10. Another option that does not require extra hyperparameters is the Parametric ReLU [61], which is a Leaky ReLU with a parameter α that is learned as part of the parameters of the model. These two alternatives can be an improvement when a model is facing the dying ReLU problem, but will not necessarily make a significant difference in case this problem isn't present.

The Scaled Exponential Linear Unit (SELU) [64] is another member of the hockey-stick family, characterized by an exponential asymptote for negative values and a linear function for positive values, as follows

$$f(x) = \lambda \begin{cases} x & \text{if} \quad x > 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{4.10}$$

The SELU activation function was proposed as part of Self-normalizing Neural Networks (SNN) [64] and has the remarkable property normalizing the distributions of outputs from all layers in the network for certain values of λ and α. Another important remark with respect to this activation function is that it has an asymptote on $-\lambda\alpha$ for negative values.

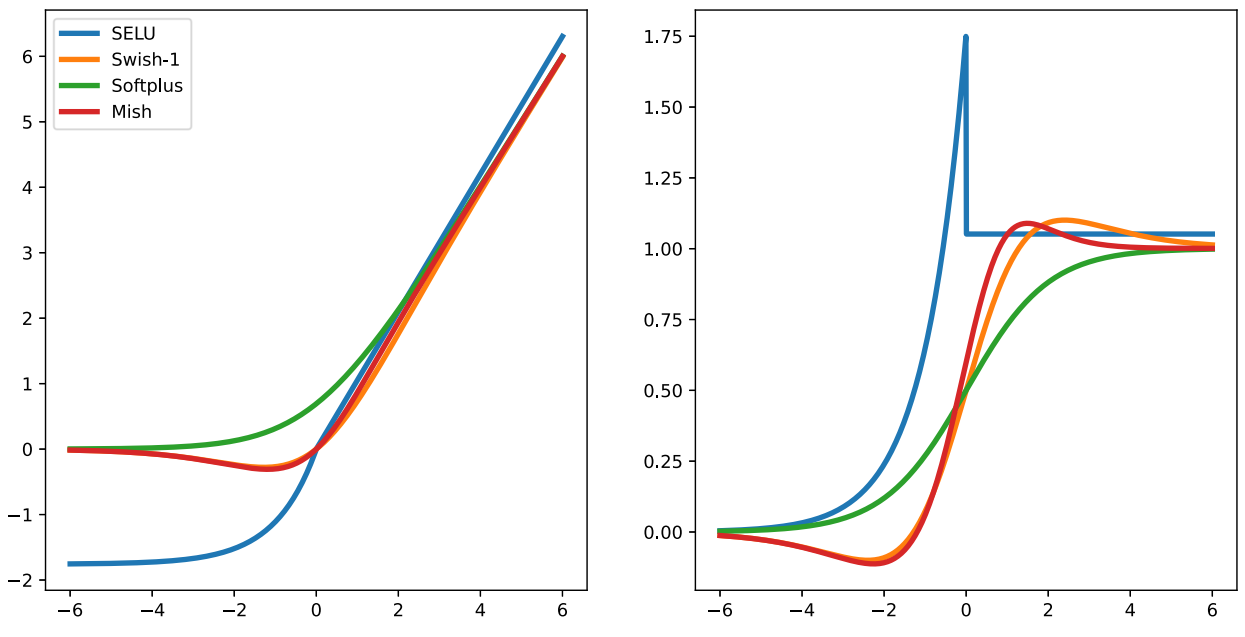


Figure 4.11: Hockey family of functions (left) and their corresponding derivatives (right).

Another alternative is the Sigmoid Linear Unit (SLU), more generally known as "Swish" [65] which consists of the function $f(x) = x\sigma(x)$. The Swish activation function is said to be self-gated, in this perspective the input is weighted by the sigmoid of itself, which as we stated before is a smooth approximation of the step function. This interpretation will be more clear once we see recurrent neural networks in section 4.6. One notable difference with other hockey-stick functions is that this function is not monotonic, implying that it has negative derivative values and a minimum around $x_0 \approx -1.27$, where the function takes a slightly negative value.

The "Mish" activation function [66] characterized by $f(x) = x\tanh(sp(x))$ claims consistent improved performance over ReLU. Inspired by Swish and found by systematic

exploration of the characteristics that made Swish perform well, it replaced the sigmoid gate for the hyperbolic tangent of the softplus. Furthermore, Mish is claimed to be self-regularized thanks to a term that appears on its derivative that behaves as a preconditioner by "smoothing" the gradients[13]. The Softplus activation function is defined as $sp(x) = \ln(1 + e^x)$, proposed as a smooth approximation of the ReLU function.

Another important nonlinearity is the softmax function. It is described by

$$f(x) = \frac{e^x}{\sum_{i=0}^{N-1} e^{x_i}} \tag{4.11}$$

and is equivalent to an exponential function normalized by its ℓ_1 norm. It is then a mapping to the positive semi plane followed by a magnitude normalization. This allows for describing probability distributions and is usually used as the output of classification models to represent the confidence of the input being in each class.

4.6 Recurrent Neural Networks

A subset of NN tailored for sequence processing is referred to as Recurrent Neural Networks (RNN). They differentiate from standard NN by having inputs to be processed sequentially and having a reference to the previous output, meaning they have feedback, allowing information from previous inputs to persist. An Elman RNN layer is described as

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{4.12}$$

this is illustrated in figure 4.12. These layers are updated through what is called "back-propagation through time" in which the recurrent layer is unfolded through time to obtain an equivalent FC NN, as depicted in figure 4.12, which can be trained through standard backpropagation.

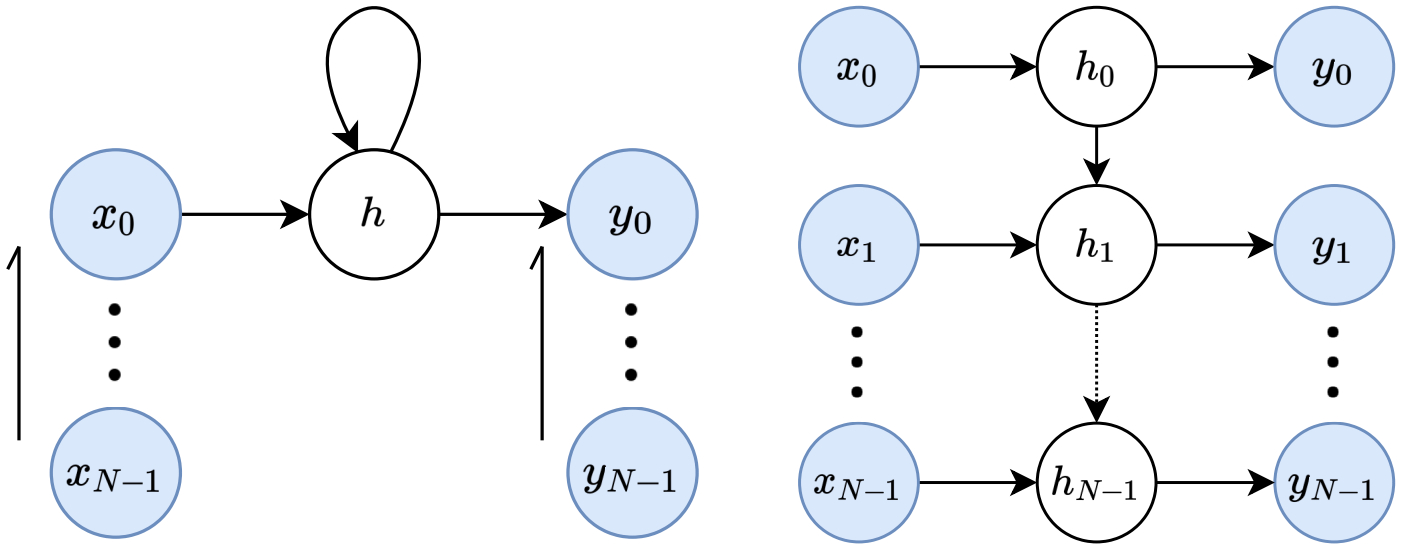


Figure 4.12: Single hidden layer RNN (left). Unfolding in time of a RNN (right).

[13]For more details on this refer to [66].

As can be seen, a relatively simple RNN can represent a very deep model, this is advantageous since this increases the approximation capacity of the model with a reduced parameter size. On the other hand, this is a problem because of an issue referred to as "vanishing/exploding gradients" that arises from the fact that for calculating the gradient of a parameter in a layer, it must be multiplied by the gradient of all previous layers, as described in section 4.4. If the gradients in each layer are small(big), then the gradient of downstream layers will be increasingly smaller(bigger) by the composite effect[14]. Furthermore, vanilla RNNs in practice are not able to handle long-term dependencies [67].

Long Short Memory Networks (LSTMs) [68] are a type of RNN that addresses these issues of vanilla RNNs. The LSTM cell is depicted in figure 4.13. The key ideas of LSTMs are, first, the addition of a cell state C_t tailored to maintain information from previous states, and second, the use of gates that mediate the flow of information. Gates are composed of a single NN layer with a sigmoid function, in this way scaling the output to be in the range $[0, 1]$. The output of these gates then multiplies the gated signals to optionally let through information.

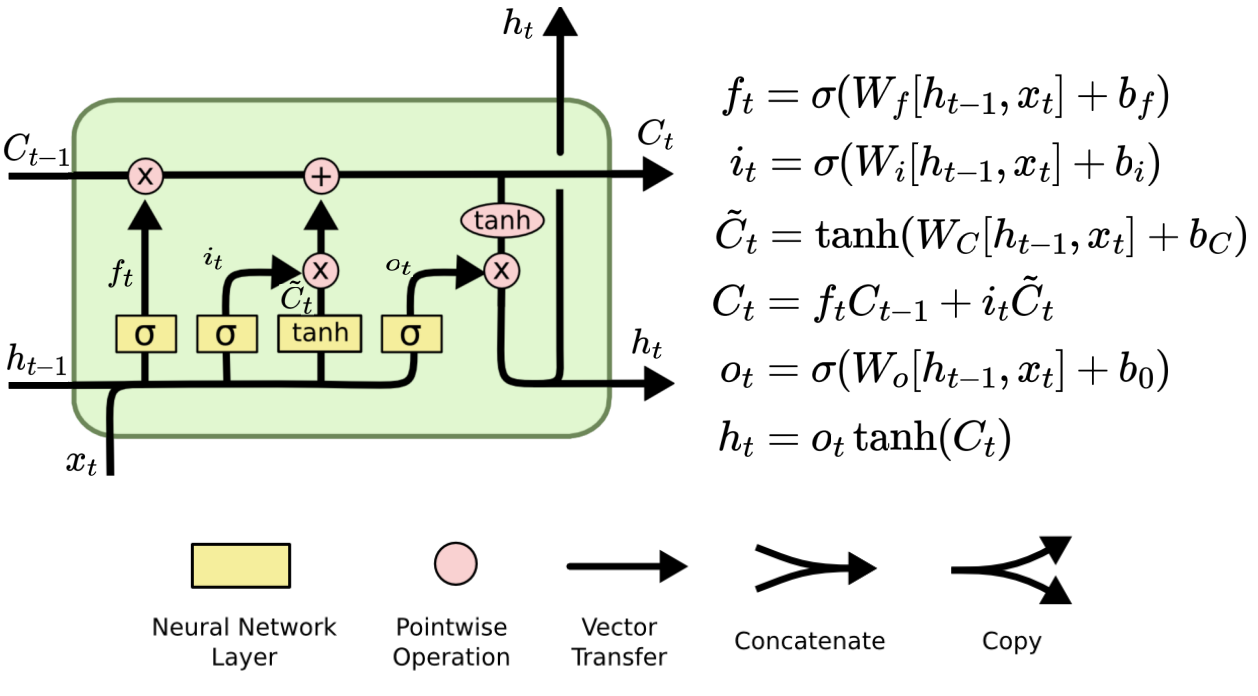
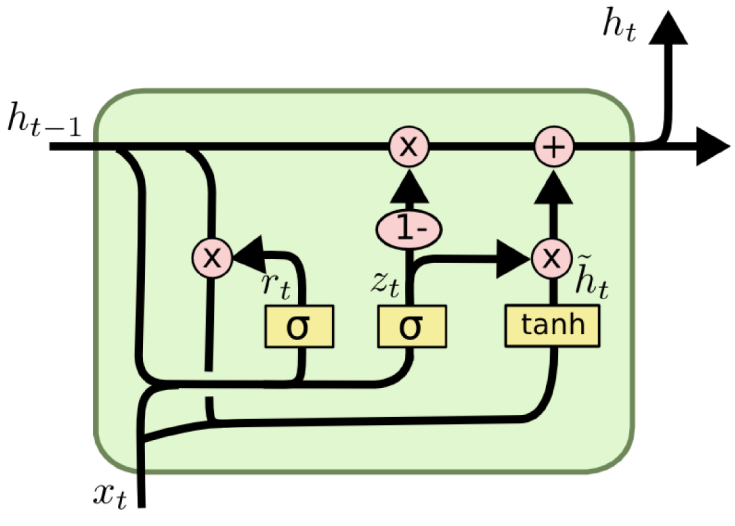


Figure 4.13: LSTM cell. Modified from [69].

An alternative to LSTM is Gated Recurrent Units (GRU). They differentiate from the LSTM by having only a single hidden state and a different gate layout, which merges the function of some gates of LSTM. Overall it is a simpler alternative. It is depicted in detail in figure 4.14. For a more in detail explanation of these types of architectures refer to [69].

[14]For a more proper definition of this problem refer to [61].



$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W[r_t h_{t-1}, x_t])$$

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

Figure 4.14: GRU cell. Taken from [69].

4.7 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are heavily inspired by the kinds of computations done in the visual perception circuits of mammals. In particular, CNN structure follows closely that of the early visual cortex, which is structured in ”columns” with almost homogeneous layers of neurons that perform an equivalent computation from upstream images but in different positions. This can be more formally described as having a spatially equivariant representation, meaning that a translation of the input will have an equivalent translation of the output. An accepted interpretation of this is that these early layers perform a feature representation of the image caught by the retinas that is efficient for downstream tasks such as classification or tracking. FC NN loses all spatial structure of the input and sees it as independent elements. To address this task in a more efficient manner, it is reasonable to hard-code translation equivarience into the model. We will describe more formally what equivariance and invariance are in section 4.10.3.

CNNs achieve translation equivariance replacing the matrix multiplication with a convolution by a kernel. A kernel is a matrix with the same number of dimensions of the input but with a reduced size, which is convolved with the input. For the 2D case, the convolution by a $K \times L$ kernel is defined as

$$x_{ij} * k = \sum_{n=0}^{K} \sum_{m=0}^{L} x_{(n+i)(m+j)} k_{nm} \tag{4.13}$$

For example for a 1D signal, a kernel $k = \{-1, 1\}$ would compute the 1st discrete differences as $y[i] = x[i + 1] - x[i]$. In the context of signals, this can be used as a filter (FIR filter) and in the context of images, this can be either used to filter out undesired noise by smoothing the image or to sharpen the image. Alternatively, it is used in classical Computer Vision (CV) as a feature extractor, for example, to detect edges or corners.

In CNNs each layer is described by a convolution and a subsequent nonlinearity with

bias as

$$y = f(W * x + b) \tag{4.14}$$

here the weight matrix W represents a set of C_{out} kernels. For the 1D case, it maps an input x of size $L \times C_{in}$ to an output y of size $L_{out} \times C_{out}$, through the convolution with W of size $C_{out} \times K \times C_{in}$ and the addition of a bias of size C_{out}. This means that each kernel weights K-sized windows of x along its full depth C_{in}. The output size L_{out} depends on K and the amount of padding P added to the input as $L_{out} = L - (K + 1)//2 + P$, where padding refers to appending values to x to increase its size. If one wants to maintain a constant length and not incur in any translation, K must be odd, the padding must be $P = (K + 1)/2$ and equal on both sides of the input. The 1D convolutional layer can be more easily understood by the calculation of the jth channel of the ith output depicted in figure 4.15 and given by

$$y_{ij} = f\left(b_j + \sum_{k=0}^{K}\sum_{c=0}^{C_{in}} W_{jkc} x_{(i+k)c}\right) \tag{4.15}$$

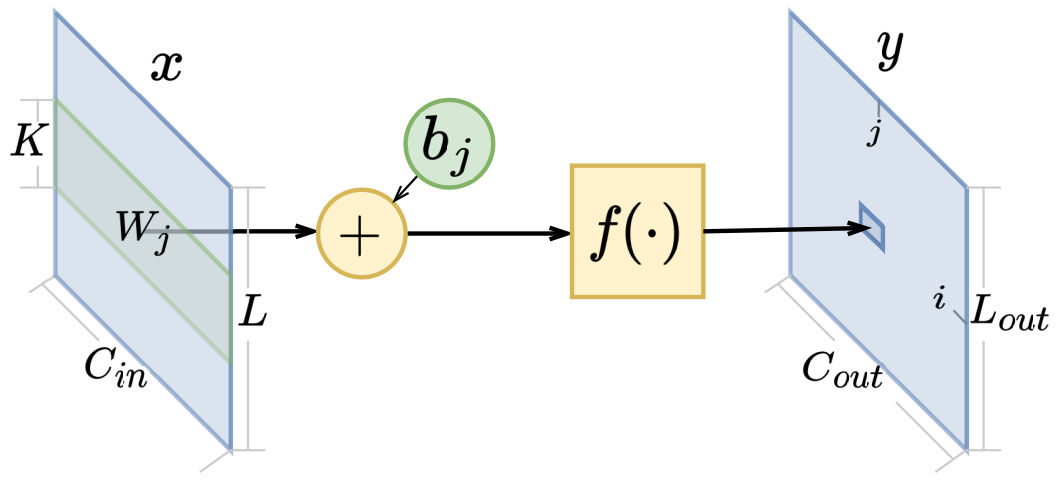


Figure 4.15: Illustration of the computation of a 1D CNN layer for the output y_{ij} depicted as a small rectangle inside the whole output y. The green rectangle represents the weighted sum of a section of the input by the kernel. The jth kernel is given by W_j which is the jth slice along the first dimension of the kernel matrix W.

For the 2D case, everything is equivalent but the input length L is replaced by the input shape $H \times W$, the output of length L_{out} is replaced by its shape $H_{out} \times W_{out}$ and the kernel length K is replaced by its shape $K_h \times K_w$. The relation between the input shape and output shape is equivalent to the 1D case but for each dimension, making the requirement to maintain the shape over kernel size and padding analogous. The 2D convolutional layer operation is depicted in figure 4.16 for a single output in position i, j, k and is given by

$$y_{ijk} = f\left(b_k + \sum_{h=0}^{K_h}\sum_{w=0}^{K_w}\sum_{c=0}^{C_{in}} W_{khwc} x_{(i+h)(j+w)c}\right) \tag{4.16}$$

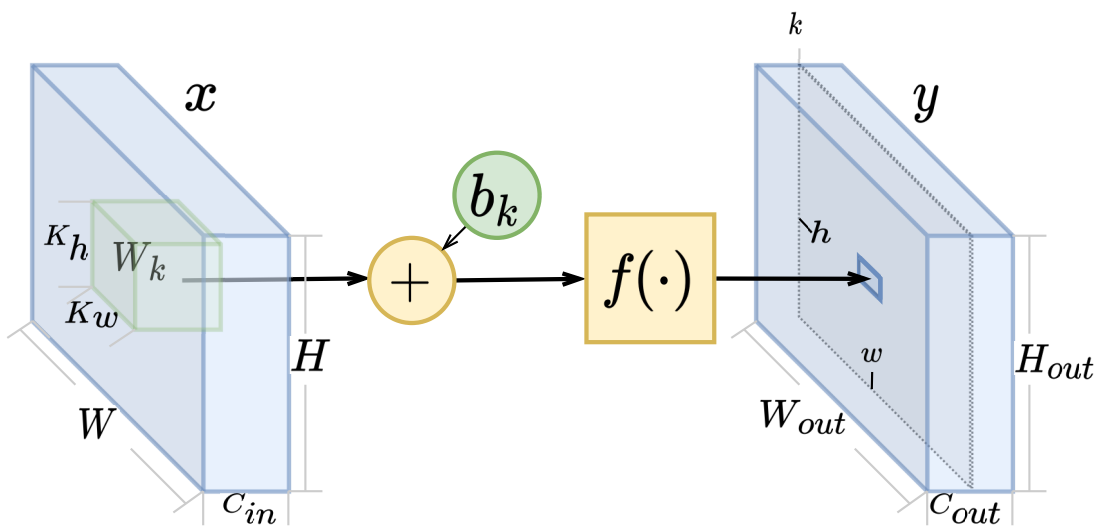


Figure 4.16: Depiction of the computation of a 2D CNN layer for the output y_{ijk} shown as a small rectangle along the kth channel plane of the whole output y. The green cube represents the weighted sum of a patch of the input by the kernel. The kth kernel is given by W_j which is the kth slice along the first dimension of W.

CNNs are usually constructed as a set of convolutional layers with an output that is subsequently flattened and fed to a FC NN. It is theorized [70] that convolutional layers function as nonlinear feature extractors that map the input space to a feature space that better represents the relevant variance of the input distribution. This improved representation can then be fed to a simple classifier or regressor, such as a FC NN, that would not be capable of achieving good performance from the original input space, especially for high dimensional data. CNNs are able to extract this feature representation by creating hierarchical representations of input data, from simple representations in early layers to increasingly more complex representations along its depth. Each layer extracts increasingly higher-level features by composing the features of the previous layer. This is depicted in figure 4.17 for the case of face detection.

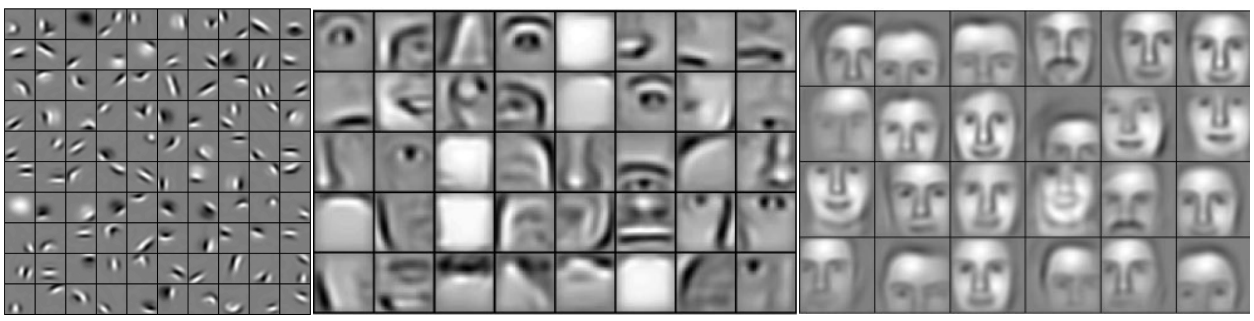


Figure 4.17: Example on the kind of features a CNN could learn in the task of detecting faces. Modified from [71].

We can also modify CNNs to be invariant to some transformations, meaning that the output is the same for an input and certain transformations of the input. For example for classification, it is desirable to have translation invariance, which is usually achieved by the use of pooling layers. Pooling layers are operators that subsample the previous layer to reduce its size. A typical pooling is the ”Max pooling” which takes the maximum value for

each channel on a window, this gives invariance to small translations. The Max pooling operator for the 1D case is described by the operation

$$y_{ic} = Max(\{x_{ic}, \ldots, x_{(i \cdot S+K)c}\}) \tag{4.17}$$

where S is the stride and K is the window size. For the 2D case it is described by the operation

$$y_{ijc} = Max\left(\begin{bmatrix} x_{i \cdot S_w j \cdot S_h c} & \cdots & x_{(i \cdot S_w + K_w) j \cdot S_w c} \\ \vdots & \ddots & \vdots \\ x_{i \cdot S_w j \cdot S_h c} & \cdots & x_{i \cdot S_w (j \cdot S_h + K_h) c} \end{bmatrix}\right) \tag{4.18}$$

where S_h and S_w are the vertical and horizontal strides and $K_h \times K_w$ is the size of the window.

Translation invariance can be achieved by using multiple pooling layers interleaved between standard convolutional layers. Other types of invariance, such as rotation and scale invariance or more generally, "warping" invariance within reasonable bounds, are usually achieved by a process called Data Augmentation, which will be described in section 4.10.3.

4.7.1 Increasing Depth

Creating deep architectures is one of the factors that gave rise to the DL renaissance, allowed by the increased capacity through GPU computing. This can be seen in their dominance, to this day, in image competitions in CV since the inception of architectures such as AlexNet [72] and VGGNet [73]. But one observed issue of CNNs is what is known as *degradation problem* [61] in which increases in depth of a model beyond a point will lead to a decrease in performance both in training and testing. This is counterintuitive as the expressive capacity of CNNs should be greater than shallower networks since any extra layer could learn the identity and be equivalent to its shallower counterpart. This phenomenon is thought to be more an issue of learning rather than of approximation capacity of the model, explained by vanishing gradients, in a similar manner to how this phenomenon affects learning in RNNs.

The standard way to address this issue is to add parallel identity paths that bypass sections (or blocks) of the model and relay the features upstream in the model. This can be seen from the feature perspective, in which upstream blocks have access to low-level features along with high-level features, which can be useful to extract higher level features. The most accepted perspective is that this allows gradients to flow back more easily and in this way, the vanishing gradient issue is addressed. This was introduced in the ResNet [74] architecture in which the residual connection sums the previous block output to the following block output. This requires the outputs to be of the same shape or to be rescaled accordingly (usually by a linear transformation). There are other alternatives such as the one introduced in the DenseNet [75] architecture in which features of previous blocks are concatenated along the channel dimension. Here the requirement is to maintain the shape of the output but there is no requirement to match channel depth. These architectures are

depicted in figure 4.18, where the convolutional blocks represent a small set of convolutional layers with pooling layers.

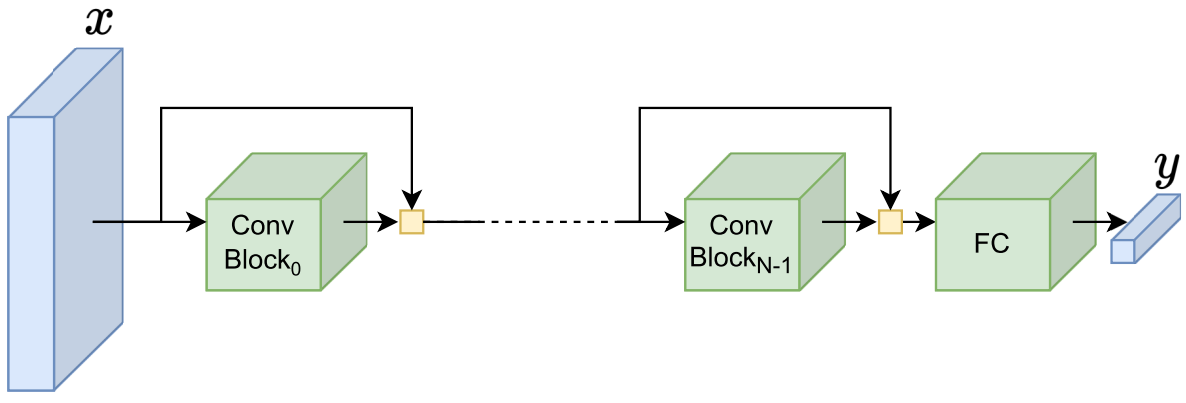


Figure 4.18: Depiction of a residual or dense CNN with N convolutional Blocks. The yellow block can represent either channel-wise concatenation for dense connections or summation for residual connections.

4.7.2 Dilated Convolutional Neural Networks

Another issue of CNNs is the increasing parameter number requirements for achieving big receptive fields (RF), described by the input span used to calculate a certain output. RF can be either increased by adding depth or by increasing the kernel span, but this can over-complify the model hindering its learning capacity. Alternatively, bigger pooling can also increase RF but lose positional precision. This is an issue when a big RF is needed. A good example of this is sequence processing, in which relevant information can be spread over a very long span. For this reason, RNNs were the standard architecture used for long-sequence processing.

An alternative to solve this issue is to use dilated convolutions, which increase the RF by having spread kernels that are made sparse through upsampling. This is illustrated in figure 4.19, where the information flow of an equivalent CNN with and without dilation is shown. This was proposed for causal sequence processing [76] and for image segmentation [77], by taking inspiration from the kind of operations made through wavelet multi-scale decomposition [78]. In this way, RF and complexity are disentangled and can be independently scaled to necessary levels.

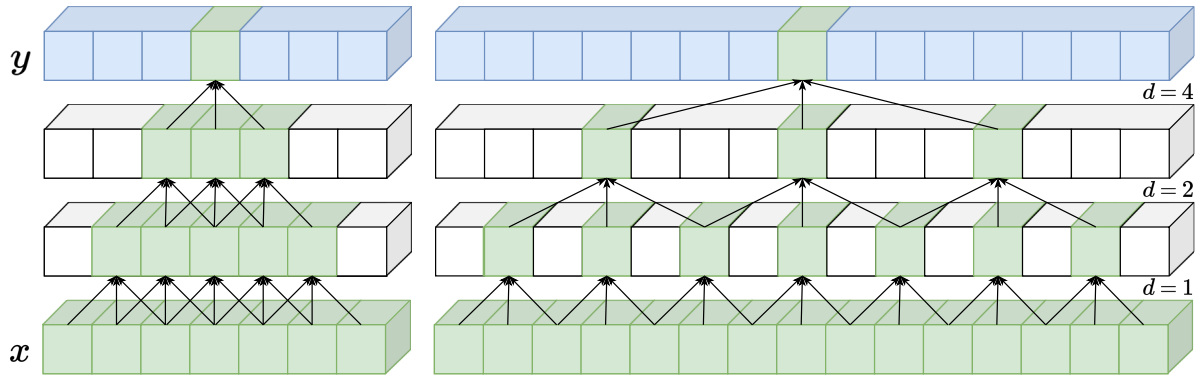


Figure 4.19: Information flow in a 3-layer 1D CNN with kernels of length $K = 3$, with (right) and without (left) dilation. d is the dilation on each kernel. The depth of blocks represents channels.

4.8 Autoencoder

In general, NNs are used for supervised learning. In this setting they require large datasets of labeled data, meaning datasets for which each input the output is known. Often, labeled datasets are not readily available and may be expensive to manufacture, especially if expertise is required for a human to do it. Usually, practitioners will manually manufacture their datasets, or alternatively use services such as *Amazon Mechanical Turk* that crowdsource such labor. An alternative to this is to have only a small labeled dataset and use an extensive unlabeled dataset to obtain an appropriate embedding space through unsupervised learning that eases the learning task on the labeled dataset.

One approach to accomplish this with NNs is the Autoencoder (AE). It relays on constructing two NNs, an Encoder and a Decoder. The encoder transforms the input space x into a feature space $\tilde{x}$, and then the decoder transforms $\tilde{x}$ into an estimation of the input $\hat{x}$. This can be trained in an end-to-end manner by minimizing $\mathcal{L}(x, \hat{x})$ where $\mathcal{L}$ is a metric of the difference between input and reconstruction. The feature space representation is made to be "efficient" by constraining it, usually through a bottleneck that reduces the dimension with respect to the input. Although, it can be constrained in other ways such as in Sparse AE, where a regularization term is added to the loss function that penalized a surrogate of the ℓ_0 of $\tilde{x}$. The concept of regularization is explained in more detail in section 4.10.1. In either case, by constraining the feature space, the autoencoder must find an embedding space that represents only essential information of the input. An example is illustrated in figure 4.20 for the case of a CNN AE.

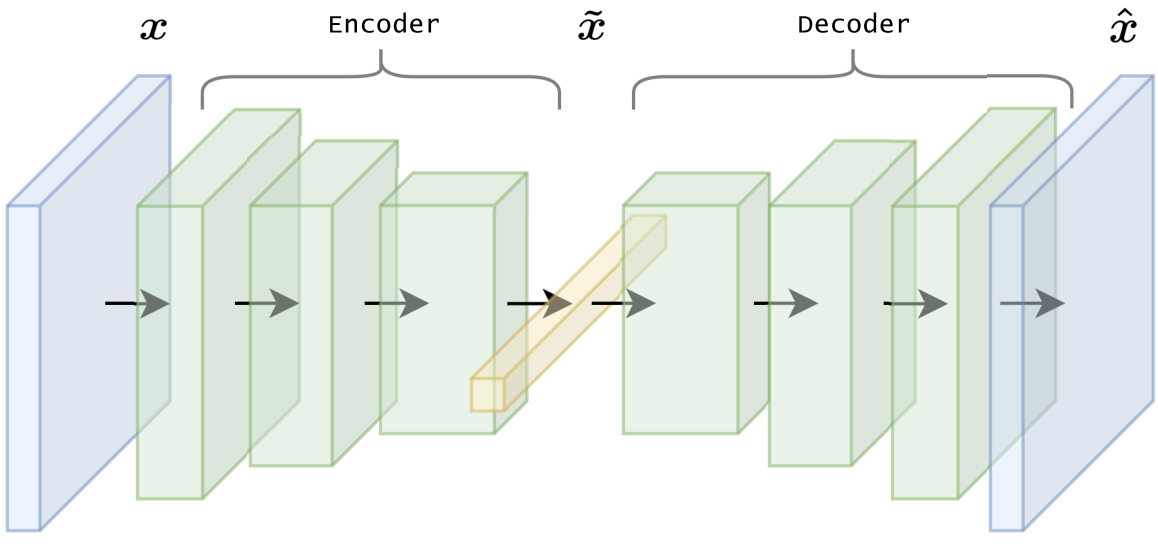


Figure 4.20: Convolutional AE.

4.9 Training

In this section, we will clarify some terminologies often used in the context of training NNs, and outline some commonly used heuristics.

4.9.1 Adaptive Optimizers

As was mentioned in section 4.4, there are optimizers that modify the vanilla SGD algorithm. SGD with momentum [79] introduces an exponential average over the parameter update. The name momentum comes from the analogy to physics, as in a ball accelerating down a slope. The algorithm is described by the following update rule

$$v_t = \gamma v_{t-1} + \eta g_t \tag{4.19}$$

$$w_t = w_{t-1} - v_t \tag{4.20}$$

were $g_t = \nabla_w J(w)$ is the gradient. Typical values of γ are in the order of $\gamma \approx 0.01$. For a better understanding of optimizers with momentum refer to [80].

Another popular alternative is Adam [81], which incorporates momentum and scaling of gradients. The essence behind the implementation of Adam is to have a scaled learning rate, to increase it for parameters with small gradients, by normalizing over the square root of the square of the gradient of recent gradients. The weights are updated as

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.21}$$

with the exponential means of the first and second momentum given by

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.22}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.23}$$

and the unbiased estimates of the first and second momentum given by

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.24}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.25}$$

ϵ is a small number to prevent dividing by zero when gradients are too small. β_1 and β_2 are the exponential decay rate for the first and second moment estimates, respectively. Default values for these are $\beta_1 \approx 0.9$ and $\beta_2 \approx 0.999$.

AdamW [82] modifies the typical implementation of weight decay of Adam by decoupling the weight decay from the gradient update. Weight decay is used to enforce ℓ_2 regularization over the parameters. It is implemented in Adam usually by adding a regularization term to the loss function as $G(w_t) = J(w_t) + \frac{\delta}{2}||w_t||_2$ so that the gradient is then computed as

$$g_t = \nabla_w G(w) = \nabla_w J(w_t) + \delta w_t \tag{4.26}$$

where δ is the rate of weight decay. In AdamW, weight decay is added directly in the parameter update as

$$w_t = w_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \delta w_{t-1} \right) \tag{4.27}$$

4.9.2 Hyperparameters

Hyperparameters [58] are the set of parameters that are not learned through the learning process. These must be chosen before the optimization process. These optimal values are not known a priori and have to be found, usually by sampling hyperparameters values to find an optimal.

The most naive approach is random search [83], where bounds for each hyperparameter are set. Then the solution space is uniformly sampled for a number of iterations or until no more improvement is observed. Another alternative is grid search [83], where the solution space is reduced to a grid equidistant along each hyperparameter range and sampled exhaustively to find the optimal value. These are sufficient for classical ML models, as they are faster to train.

For more expensive to train models, such as DL models, other methods that sample in a more ”intelligent” manner are used [84]. These are usually based on bayesian optimization [85], which consists in constructing a surrogate function that approximates the function we want to optimize. Usually, a Gaussian Process Regressor (GPR) is updated iteratively by sampling points given by an acquisition function. A GPR not only approximates the function but also gives information about the uncertainty of the approximation at each point. The acquisition function uses this information to sample the next point by a trade-off between the ”exploitation” of known minimal neighborhoods and the ”exploration” of neighborhoods that minimize uncertainty.

4.9.3 Overfitting

In the previous section we described strategies to improve the performance of the model. More concretely, what this refers to is for the learned model to have a low loss and be able to generalize, meaning that it can learn from a distribution of data and maintain a relatively good performance on out-of-distribution data. In ML, a model is said to generalize well when it is neither overfitted nor underfitted, loosely referring to having a model that is neither too complex nor too simple. This can be illustrated for a classification problem in figure 4.21. As can be seen from the figure, an overfitted model matches all data points in the dataset but probably is not able to maintain that performance if another sample is taken. On the other hand, the underfitted model probably will maintain the out-of-distribution performance but still has improvement potential.

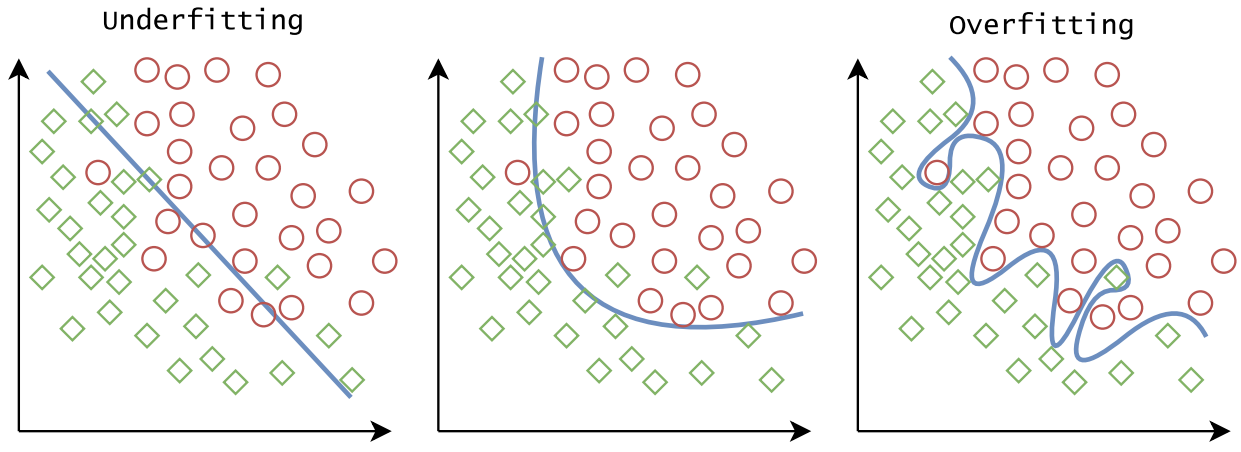


Figure 4.21: Underfit, adequately fit and overfit models for a classification problem.

In ML, most classical models have convex formulations, ensuring an optimal fit of the model. The hyperparameters of the model that better generalize are found through a process called cross-validation, in which a set of the training data referred to as test data is left out and used to measure performance. The set of hyperparameters that best performs in this subset is selected. The most common cross-validation scheme in ML is known as k-fold cross-validation, in which the data is partitioned into k subsets called ”folds” and for each set of hyperparameters the model is trained over all $k-1$ fold subsets and tested on the left out fold. The performance is then the mean over all the cross-validations.

In comparison to classical ML algorithms, DL generally requires greater computing power. This translates into greater learning times, which makes standard k-fold cross-validation prohibitively expensive. Instead, a subset of data is set as test data and another subset as validation data, both usually in the order of 10% of the data. The remaining portion is the training data and is used to train the model. The test data is used for cross-validation to find the optimal hyperparameters, and after this, the validation set is used to validate the out-of-distribution performance.

Another significant difference between DL models with respect to classical ML models is that the former are usually trained through gradient descent. Since the loss landspace is non-convex and different for each subsample of data, the optimal over the training dataset

will overfit to that subset of data, leading to poor generalization over the test set. In practice, the loss over both sets initially improves in conjunction until the model starts to overfit, leading to a progressive decrease of the loss over the test set. This is illustrated in figure 4.22. To address this, the model is trained over the train set, while the test set performance is monitored. The model is trained for a preset number of epochs or until the test set performance stops improving. Then the model parameters where the test set performance is optimal are selected as optimal parameters. This can be understood as setting the number of epochs as a hyperparameter.

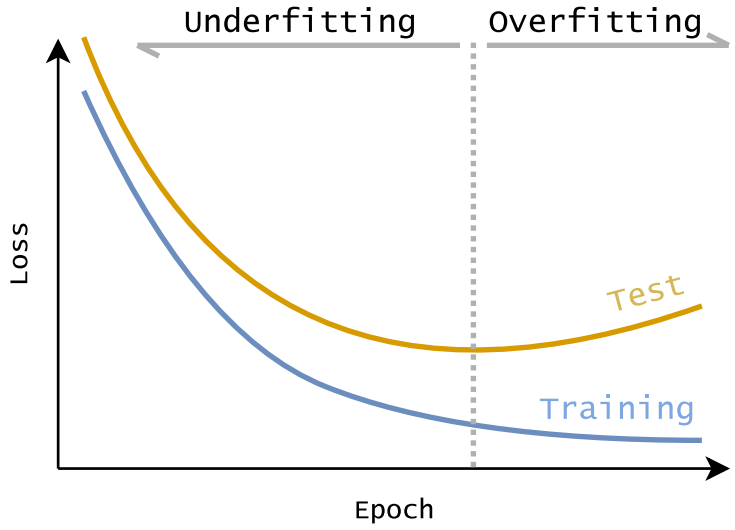


Figure 4.22: Typical loss curves for the train and test set.

4.9.4 Learning Rate Schedulers

The Learning Rate (LR) is an especially significant hyperparameter because it not only impacts the final performance of the trained model but also the required number of epochs, as it mediates the rate of convergence. If the LR is too small, it will take too long and may get stuck in local minima or saddle points. On the other hand, if it is too big, it may overshoot the optimal value valley and fail to converge or even diverge. Generally, choosing a sufficiently adequate range of LR is a task that the practitioner must find by naive trial and error around some typical values, usually by looking at the convergence rate over a few epochs.

To address this problem, a commonly used heuristic is to use LR schedulers. They consist in varying the LR rate throughout the learning schedule. Most are tailored to allow a high learning rate at the beginning of training, and progressively decaying the LR to be able to converge when arriving at the minima. Some examples of strategies following this direction are `StepLR` described by

$$lr_{epoch} = \begin{cases} \gamma lr_{epoch-1} & \text{if } epoch \% step\_size = 0 \\ lr_{epoch-1} & \text{otherwise} \end{cases} \tag{4.28}$$

where γ is the decay rate and $step\_size$ is the number of steps between each decay. Another

example is `ExponentialLR` described by

$$lr_{epoch} = \gamma lr_{epoch-1} \tag{4.29}$$

which is equivalent to `StepLR` with $step\_size = 1$.

Another alternative is to decrease the LR when training stagnates, by monitoring a metric of performance over the test or train set. This strategy is known as `ReduceLRonplateau`, and consists in decaying the LR by a decay rate γ whenever a certain number of epochs do not improve the monitored metric.

More recently, a set of LR schedulers known as cyclical LR schedulers have been proposed [86]. They consist of multiple phases of increasing and decreasing learning rate between a minimum value and a maximum value, following different curves such as triangular, exponential and cosine. Some variations decrease the maximum value on each new cycle. This set of LR schedulers has been extensively used and proven to give good results. The intuition behind their performance is that they allow traversing multiple valleys in the parameter loss landscape, allowing in this way to reach flatter valleys that should generalize better. This is illustrated in 4.23.

Following the success of these LR schedulers, OneCycle LR scheduler [87] was proposed, claiming convergence in a significantly reduced number of epochs. It consists of a single cycle of a typical cyclical LR scheduler, using an exceptionally high LR at the top of the cycle. Similarly to the intuition behind cyclical LR schedulers, this is theorized to allow traversing multiple valleys before decaying the LR.

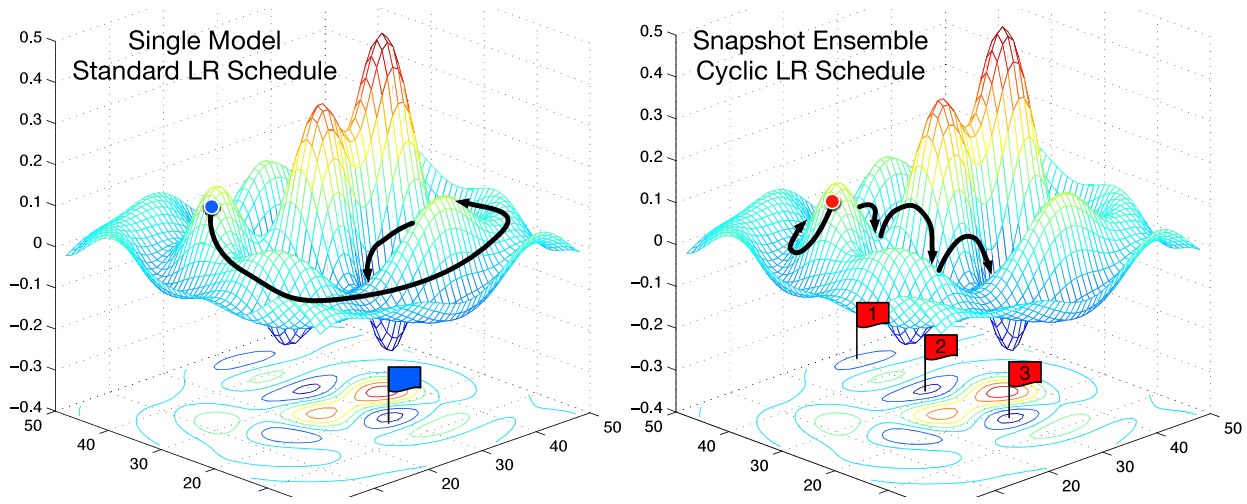


Figure 4.23: Taken from [88].

4.9.5 Learning Rate Finder

LR is usually one of the most important hyperparameters in the sense that it can have a big impact on improving the performance of the model. Adaptive optimizers, as described in section 4.9.1, relax this by making default optimizer hyperparameters sufficiently good for most applications. If performance improvement is required, a more suitable learning rate

can be found as described in section 4.9.2.

Another recent alternative known as Learning Rate Finder (LRFinder) [89] proposes a heuristic for finding adequate values of LR. It consists of training over the dataset once, while increasing the learning rate over a preset range for each batch. This procedure generates a loss curve with a shape similar to the one in figure 4.24. The adequate LR values are the ones that achieve the biggest improvement per step, as quantified by the derivative.

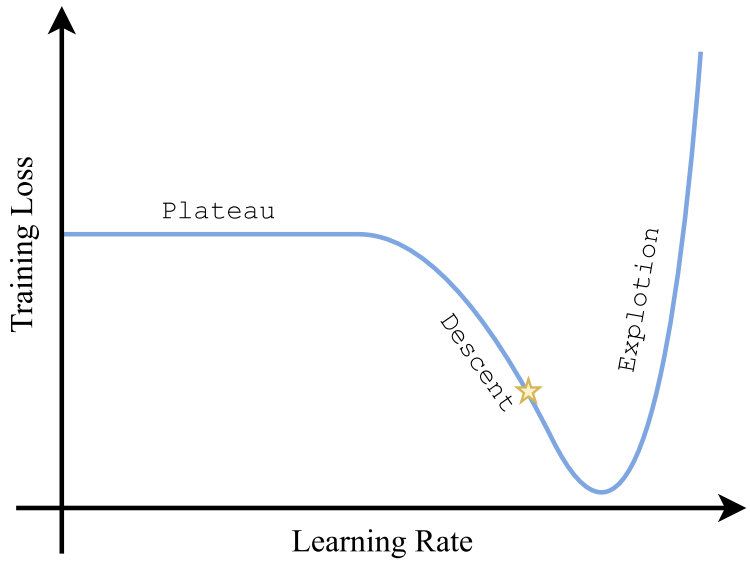


Figure 4.24: LRFinder typical curve. The star represents an adequate value for LR according to this method.

4.10 Inductive Bias

In the context of learning algorithms, Inductive Bias is the set of assumptions (explicit or implicit) in the learning algorithms. In the context of DL, these can be either by tailoring the structure of the model or by tailoring the loss function to impose assumptions known about the phenomena being learned. This must be chosen carefully to not hinder the expressivity of the model, but if well done it can help to better convergence and for improved generalization.

4.10.1 Regularization

The more straightforward way to add inductive bias to a model consists in adding additional terms to the loss function, which stirs the algorithm into a desired secondary objective, this is known as Regularization. A typical example is weight regularization, where usually an objective of sparseness is sought, as quantified by the ℓ_0 pseudo-norm[15], where the ℓ_p norm is defined as

[15]The ℓ_0 is not a proper norm since it does not have the absolute scalability property.

$$||X||_p \triangleq \sum_{x \in X} |x|^p \tag{4.30}$$

The ℓ_0 is then the number of elements different than zero. This pseudo-norm is ill-posed for gradient-based algorithms, so usually the ℓ_1 norm or the ℓ_2 norm are used as surrogates. As mentioned in section 4.9.1, ℓ_2 regularization is referred to as weight decay. This is because the derivative is the magnitude of the weights and, in consequence, this regularization in practice is a progressive proportional shrinking of weights along the training. These regularizations implicitly assume that sparser connections will generalize better. This is referred to as applying Occam's razor principle [58] by choosing the simplest solution that closely matches the training data. Other regularizations can be used, for example over the layer activations, as we briefly mentioned for the case of Sparse AE in section 4.8.

4.10.2 Parameter Constraints

In the previous section, we saw how we could impose soft constraints over the weights, in order to make them sparse. We can also impose hard constraints over a set of parameters. This may be useful, for example, when building a structured NN that models physical phenomena, where the parameters have known constraints. Another example is in the context of RNNs, where it has been proposed to control the singular values of the weight matrices to be well-conditioned, in order to improve convergence.

If we want to constrain the parameters to be in $w \in \mathcal{W} \Subset \mathbb{R}^d$, where d is the number of parameters, the more naive approach is called projected gradient. It consists in performing a gradient descent step in the unconstrained space and then constraining the result by a projection to the constrained space. This can be formally expressed for the standard SGD optimization as

$$w_{t+1} = Proj_{\mathcal{W}}(w_t - \eta \nabla_w J(w_t)) \tag{4.31}$$

If $\mathcal{W}$ is a box constraint, then the projection is made by clipping the values along each dimension.

Another alternative is through reparametrization. Here the parameter w is expressed in terms of an unconstrained parameter u as $w = f(u)$, with a nonlinearity $f : \mathbb{R} \to \mathcal{W}$ differentiable and bijective. For example, for $\mathcal{W} = [0, 1]^d$ we could use the sigmoid function $\sigma(\cdot)$ described in section 4.5. Then we optimize the underlying variable u. One issue with this approach is that it re-scales the gradient, making them very small in case the parameter is in a section of the nonlinearity that has small slope. We can address this problem by using natural gradients [90], which essentially means updating u using the gradient with respect to w. For more details on these approaches to parameter constraints refer to [91].

4.10.3 Equivariance and Invariance

A function is said to be $\mathfrak{S}$-equivariant if its output is transformed in an equivalent manner for any transformation $\mathfrak{g} \in \mathfrak{S}$ of its input. This is formally stated as

$$f(\mathfrak{g}(x)) = \mathfrak{g}(f(x)) \quad \forall x \in \mathcal{X}, \mathfrak{g} \in \mathfrak{S}, \mathfrak{g}' \in \mathfrak{S}' \mid \mathfrak{g}' \approx \mathfrak{g} \tag{4.32}$$

On the other hand, a function is said to be $\mathfrak{S}$-invariant if its output is the same for any transformation $\mathfrak{g} \in \mathfrak{S}$ of its input. This is formally stated as

$$f(\mathfrak{g}(x)) = f(x) \quad \forall x \in \mathcal{X}, \mathfrak{g} \in \mathfrak{S} \tag{4.33}$$

This is especially useful for modalities that have a geometric structure and that maintain a semantic representation under certain perturbations for classification tasks, or that have equivalent perturbations of the semantic representation for regression or segmentation tasks. This is easily illustrable in the image domain where we know that the semantic meaning of an object is maintained for many transformations and that a change in the position of an object has an equivalent position change in its representation. As we saw in section 4.7, some can be integrated into the structure of the NN in the case of CNNs.

Data Augmentation

Another way to integrate invariances or equivariances into a model is through Data Augmentation. It consists in synthetically increasing the size of the training dataset by doing the kind of transformations we want our model to be invariant or equivariant to. The difference between invariance and equivariance data augmentation is that in the second case the output is transformed equivalently. Some typical examples used in image are translation, rotation, scaling, partial occlusion and more recently *Mixup* [92], where a weighted combination of images and an equivalent weighting of the target is done. In the audio domain, some examples are pitch change, rate change, background noise, time translation, and filtering.

4.10.4 Encoding

Encoding refers to a mapping of data to another representation. In the context of ML, this usually means using a transformation that yields a representation that is more useful to the downstream task. For example, when we have outputs in categorical data usually it is transformed to One-Hot encoding, as described in section 4.1. An example of input encoding is in audio tasks where usually time-frequency representations are used such as the one yielded by the Fast Fourier Transform (FFT). One notable example is Word embeddings [93] in the context of Natural Language Processing (NLP), where words are encoded in a space where semantic relations between words are equivalent to distance and direction in euclidean space.

Chapter 5

Proposed Model

As it can be stated from chapter 3, there is a large selection of algorithms that have been used to solve the problem of SWDM. A subset of these corresponds to DL models, that although tried probably because of their current popularity, do live up to their expectations, achieving good results with low inference times. In this Thesis, the intent is to develop further this approach, by considering a key objective beyond just the accuracy of the prediction, namely, its robustness to simulation inaccuracy. This is, if the simulation model has slight inaccuracies or a different distribution of noise, then a degradation of the predictions is expected, but this degradation is, in general, not fully assessed.

A naive approach to solving this would be pre-training a model on simulated data and then finetuning it on a small dataset taken from real measurements. Although this approach would probably be a good solution, it is labor-intensive as it requires real measurements to be taken. The essence of our proposal aims at solving this subproblem, how can a model be pre-trained on simulated data and then be finetuned with real data in a fully unsupervised manner?

This problem is known in DL as unsupervised domain adaptation [94]. In this scenario, a labeled source dataset is available from which the intent is to learn and then transfer that learning to an unlabeled target dataset, with the aim of improving the performance that would be obtained on the target dataset without adapting the model. This is also known as transductive transfer learning [95], the terminology used more broadly in the ML community.

In this chapter, the proposed model that addresses this problem will be presented. The architecture will be described, together with the reasons behind all design decisions, as well as the training procedure. The proposed model will be evaluated in comparison to previous methods and together with its adaptation capacity to data with different characteristics.

5.1 Architecture

This work proposes using an autoencoder (AE) to achieve unsupervised domain adaptation in the context of SWDM. As described in section 4.8, an AE is typically used for transfer learning. Instead, with the objective of unsupervised domain adaptation, we propose to first train the encoder in a supervised manner and then fine-tune it in an unsupervised manner by minimizing the reconstruction loss of the AE. The objective is to *pretrain* the encoder with simulated data and then *finetune* it with experimentally measured spectra, without requiring the FBG spectral positions. However, there is no guarantee that the resulting representation will improve the performance because the latent variable may not maintain a semantic relationship with its previous representation.

To ensure that the latent variable maintains a meaningful relationship with its previous representation, it is necessary to introduce certain constraints into the model. These constraints, known as inductive biases (see section 4.10), are designed to help maintain continuity between tasks in the latent variable. To achieve this, we encode the latent variable $\tilde{y}$ as a per-sensor Dirac delta, positioned at the corresponding peak wavelength, as illustrated in figure 5.2. To secure that the architecture achieves this implicit encoding, the following architectural features are implemented:

(i) The encoder is a CNN without pooling layers, so that the internal representations maintain the length of the input. Making the latent variable $\tilde{y}$ be translation-equivariant with respect to the input. This property is a natural feature of CNNs that is often lost when pooling layers are used to achieve translation-invariance (see section 4.10.3).

(ii) The last layer of the encoder has one channel per FBG sensor. Making the latent variable $\tilde{y}$ an array of length N with Q channels (depth), where Q is the number of FBG sensors and N is the number of spectral points.

(iii) The last activation function of the encoder is a softmax (see section 4.5), so that the latent variable $\tilde{y}$ is positive and with per-channel normalized ℓ_1-norm.

(iv) The center of mass along each channel of $\tilde{y}$ is calculated to obtain the target inference $\hat{y}$ with the FBG peak positions.

(v) A regularization is designed for the latent variable to make each channel narrow as a Dirac delta. This also functions as the information bottleneck of the AE, making the model a Sparse-AE (see section 4.8).

(vi) The decoder contains a convolution layer that learns the individual spectral profiles and positions them with the Dirac deltas, yielding the separated spectra $\tilde{x}$ (see figure 5.2).

(vii) The separated spectra and a learned transmissivity vector parameter (that characterizes the transmissivity of the fiber sections between FBGs) are used to compute the joint spectrum $\hat{x}$, following a topology-dependent computation, as described in section 2.9.1.

A schematic of the proposed model is shown in figure 5.1, and a more detailed schematic of the decoder is shown in figure 5.2, which will be further described in section 5.1.4. It can be seen from figure 5.2 how an instance with partial overlap would ideally be transformed along the model.

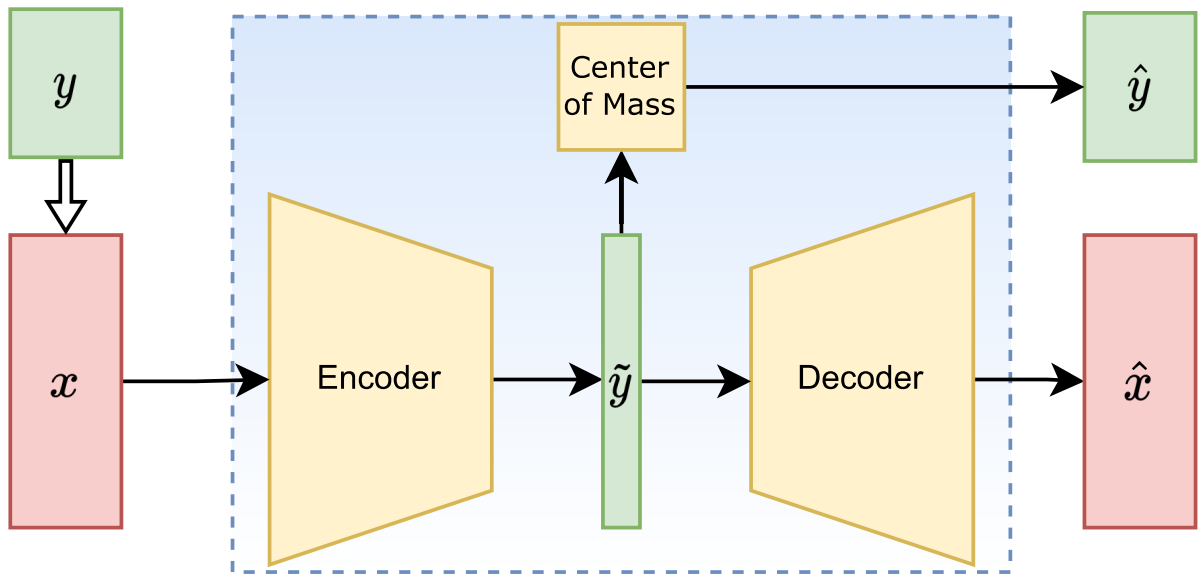


Figure 5.1: Schematic of the proposed model represented in blue, the submodules are represented in yellow. x, y are the input and the target, $\tilde{y}$ is the latent representation and $\hat{y}$ is the estimated target, $\hat{x}$ is the input estimation.

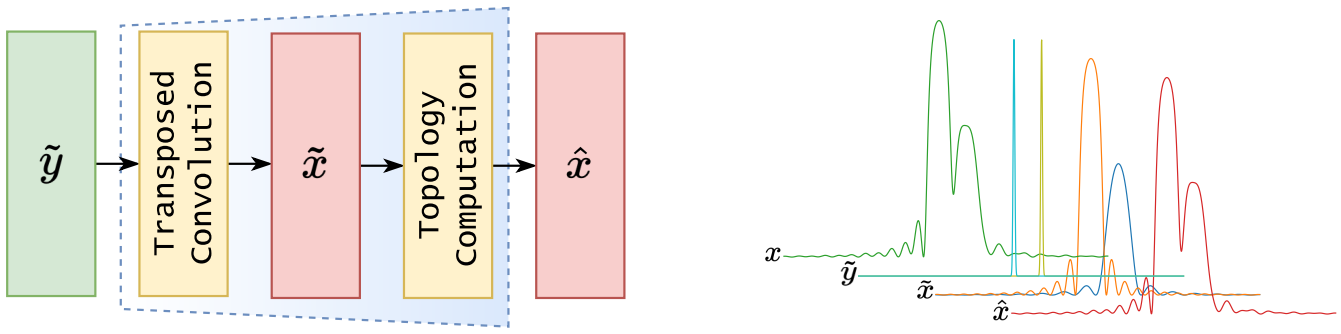


Figure 5.2: Schematic of the decoder and its submodules (left). $\tilde{y}$ is the latent representation, $\tilde{x}$ is an estimation of the separated spectra and $\hat{x}$ is an estimation of the input spectrum x. The input and the ideal internal representations intended for the model to extract (right) are depicted for the case of a 2-FBG serial topology instance with partial overlap.

The model is trained through gradient descent (see section 4.9). The loss function has a multi-objective for regression of the target, reconstruction of the input, and sparseness of the latent representation. The parameters of the model that meet this objective are obtained by minimizing the following loss function

$$\mathcal{L}(x, y) = (1 - \gamma)\mathcal{L}_1(y, \hat{y}) + \gamma\,\mathcal{L}_2(x, \hat{x}) + \alpha\,\Omega(\tilde{y}) + \omega\|\theta\|_2 \tag{5.1}$$

where $\mathcal{L}_1(\cdot,\cdot)$ and $\mathcal{L}_2(\cdot,\cdot)$ are the regression loss and the reconstruction loss, $\Omega(\cdot)$ is the sparseness regularization and $\|\cdot\|_2$ is the ℓ_2 weight regularization. The parameter $\gamma \in [0, 1]$ controls the trade-off between the two losses. The parameter $\alpha \in \mathbb{R}^+$ controls the strength of the sparse regularization and $\omega \in \mathbb{R}^+$ is the weight decay. The hyperparameters ω and α

are selected accordingly as explained in section 4.9.2. The training of the architectures is described in greater detail later in section 5.2

5.1.1 Data Generation

The data is generated as described in section 2.9 by first computing the individual spectrum for each sensor and then from this result calculating the joint spectra, which is dependent on the type of topology of the array as described in 2.9.1. The general parameters of the simulation are described in table 5.1 and the parameters corresponding to the sensors are described in table 5.2.

Table 5.1: General simulation parameters.

			Description
Q	2	3	Number of sensors.
topology	'serial'	'serial_rec'	Topology of the sensor Array.
simulation	'true'	'true'	Simulation to use for each sensor spectra.
N	2000	2000	Number of spectral points.
M	10000	30^3	Number of sampling points.
λ_0	1550[nm]	1550[nm]	Central wavelength.
Δn_{dc}	0	0	DC component of refractive index change.
Δ	2[nm]	2[nm]	Half wavelength range.
portion	0.6	0.6	Portion of wavelength range were spectral position can vary.
a	6[μm]	6[μm]	Radius of the fibre core.
n_1	1.48	1.48	Refractive index of the core.
n_2	1.478	1.478	Refractive index of the cladding.

Table 5.2: FBG Simulation Parameters

			Description
Q	2	3	Number of sensors.
A	$\{1, 1\}$	$\{1, 1, 1\}$	Transmissivity of the section of fiber before each sensor.
$\Delta\lambda$	$\{0.2, 0.2\}[nm]$	$\{0.2, 0.2, 0.2\}$[nm]	FWHN bandwidth of each sensor.
I	$\{0.5, 0.9\}$	$\{0.5, 0.7, 0.9\}$	Peak reflectance of each sensor.
topology	'serial'	'serial_rec'	Topology of the sensor array.
simulation	'true'	'true'	Simulation to use for each sensor spectra.

After generating the dataset, it is normalized, as is standard in ML algorithms. The targets are normalized to be in the range $[-1, 1]$ by scaling them as $y = \frac{\lambda_B - \lambda_0}{\Delta}$, where λ_B is the Bragg's wavelength, λ_0 the central wavelength and Δ is half the wavelength range. It is chosen to not normalize the reflection spectra, as the value is already bounded in the

range $[0, 1]$ since it represents a reflection proportion of a passive element, which only has possible physical values in that range. Moreover, it would be a complication for the decoder model. This will be discussed in detail later in section 5.1.4.

5.1.2 Encoder

As it was seen in section 3.2.3, multiple DL approaches have been tried in this context, such as RNNs and CNNs. The focus of this work is on using CNNs, as this is proposed as requisite for the intended representation it should form. Furthermore, CNNs are better suited for parallelized training, making their training on GPUs more time efficient.

Families of architectures are created, parametrized by a small number of parameters, making these a set of extra hyperparameters. These hyperparamters sets are sampled randomly or through bayesian optimization, to find a suitable instance of hyperparameters that achieve the desired performance. This design principle is inspired by the work presented in [96], which postulates a paradigm shift from the traditional approach of designing a single architecture to designing network design spaces.

The general architecture structure that is followed is shown in figure 5.3 and consists of a body, aimed at feature extraction, that consists of convolutional blocks. It is succeeded by a head of channel-wise convolutional layers[16], aimed at combining the local information of the extracted features. All body and head convolutional layers are followed by a nonlinear function. The essential difference between body and head is that the body has a large Receptive field (RF) while the head has a unitary RF. After this, there is a final channel-wise convolutional layer with Q output channels, this is, one per FBG, followed by per channel softmax nonlinearity, that generates a positive unitary ℓ_1-norm vector per channel (see section 4.5). The output $\hat{y}$ is obtained from the center of mass of each channel of $\tilde{y}$. This is done by computing the matrix multiplication $\tilde{y}v$ where v is an N-sized vector, yielding a Q-sized vector. The vector v is constructed from N evenly spaced values over the interval $(-1, 1)$, corresponding to the range of the normalized target.

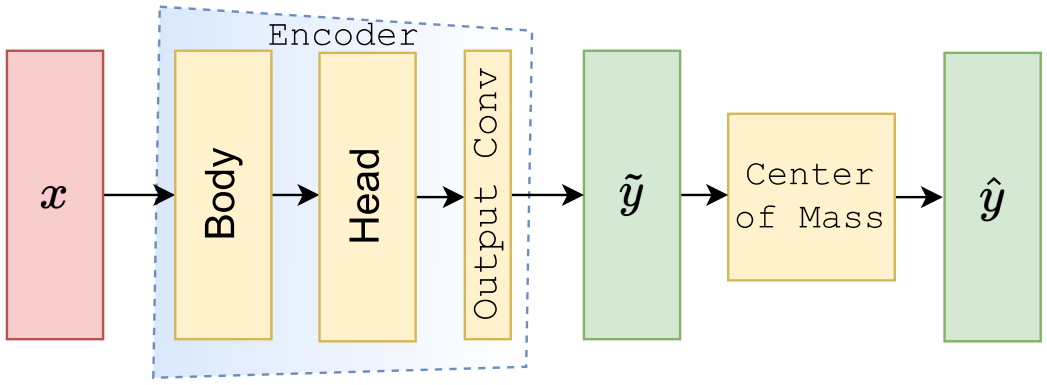


Figure 5.3: General architecture of the encoder model.

The variations between the different families are solely on the body of the network, in particular, the contents of a convolutional block and how they are procedurally constructed.

[16] This is equivalent to a 1x1 convolutional layer in the context of 2D convolutional layers.

In preliminary experiments the `dense_encoder` and the `residual_encoder` families were implemented, inspired by the DenseNet [75] and the ResNet [74] architectures. Although they gave adequate results, they are particularly limited in their ability to generate large RF, forcing the use of either large kernels or a large number of layers, both increasing the number of parameters and the computational cost. This is a common problem of CNNs often seen when dealing with timeseries signals as they often contain a large density of points, this is classically resolved by using RNNs instead, as discussed in section 4.6. Another alternative to solve this is to use Dilated convolutions (see section 4.7.2), and this is the approach taken in this work. The performance of this kind of model has already been validated for the problem of SWDM [55] and displays promising results.

Dilated Encoder Architecture Family

The `dilated_encoder` architecture family is designed to have a body consisting of dilated convolutional layers, each followed by a mish [66] activation function (see section 4.5). This activation function is chosen since it has self-regularization properties and avoids the dying ReLU problem [97] often associated with that activation function. It is avoided to use Batch Normalization [98], a technique extensively used in CNNs to aid training by normalizing neuron outputs over a batch, as it comes at the cost of obtaining different results on training and at inference [99], which could be an issue when finetuning.

The network architecture is parameterized by the number of body layers (`num_layers`), the number of head layers (`num_head_layers`), the RF (`receptive_field`), the base-2 logarithm of the number of initial channels (`init_channels_exp`) and the multiplicative growth between layer and layer (`channels_growdth`).

The dilation of a layer is computed as `dilation = dilation*(kernel-1)`, where `kernel` is the size of the kernel and with `dilation` initialized to unity. In this way, each layer tessellates perfectly with the previous layer with overlap only on the edges. The RF is achieved by setting all kernels to 3 and then progressively increasing the first kernels to reach the desired RF. The procedure that achieves this is depicted in algorithm 3, where n_{layer} corresponds to `num_layers` and R_{target} to `receptive_field`.

The design space is explored as depicted in section 4.9.2, first by manual exploration, and then when a range can be set for each hyperparameter, bayesian optimization is utilized to progressively shrink the solution space. Through this process, an adequate configuration of `num_layers=6`, `num_head_layers=1`, `receptive_field=1.0` (this value is represented in nanometers and then transformed to the number of points), `init_channels_exp=6` and `channels_growdth = 0.5` is obtained.

5.1.3 Latent Representation

As stated before, sparseness is imposed over $\tilde{y}$ as a bottleneck in order to force the encoder to efficiently encode the information of the input. This goal is 2-fold as it is also needed

Algorithm 3 Receptive Field Algorithm

Require: n_{layers}, R_{target}
  $K \leftarrow fill(shape = n_{layers}, value = 3)$ ▷ n_{layers}-sized vector
  while $receptive\_field(K) < R_{target}$ **do**
    $K[0] \leftarrow (K[0] + 1) \times 2 - 1$
  end while
  $R \leftarrow receptive\_field(K)$
  $K_{best}, R_{best} \leftarrow K, R$
  loop
    for $i \in [1, \cdots, n_{layers}]$ **do**
      $K[0] \leftarrow (K[0] + 1)//2 - 1$
      $K[i] \leftarrow K[i] + 2$
      $R \leftarrow receptive\_field(K)$
      if $K[0] < K[1]$ **then**
        return K_{best}
      end if
      if $|R - R_{target}| < |R_{best} - R_{target}|$ **then**
        $K_{best}, R_{best} \leftarrow K, R$
      end if
    end for
  end loop

for $\tilde{y}$ to be narrow, in order to approximate a Dirac delta as required by the decoder. This objective is met by adding an adequate regularization term to the loss function.

Sparseness is quantified by the ℓ_0 norm but this criterion is, in general, ill-posed for gradient-based methods, instead ℓ_1 and ℓ_2 norms are used as surrogates. The ℓ_1-norm penalizes all values equally while ℓ_2-norm penalizes proportionally to the values. This leads ℓ_2-norm regularization to leave values close to zero but not exactly zero. These norms are usually used as regularization over the set of parameters of NNs (see section 4.10.1).

Another option is to have a penalty with a parameter that sets an objective of sparseness. One option to achieve this is the Kullback-Leibler (KL) divergence between the distribution of neuron outputs and a Bernoulli distribution with mean ρ [100]. The Bernoulli distribution is a binary discrete distribution with the proportion of positive values given by its mean ρ, making it suitable for our purpose of setting a portion of the values to zero. The Kullback-Leibler penalty is described by

$$KL(\rho||\hat{\rho}) = \rho \log \frac{\rho}{\hat{\rho}} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}} \tag{5.2}$$

where ρ is the target mean and $\hat{\rho}$ is the neuron mean. This function is minimized when $\rho = \hat{\rho}$ and increases monotonically when there is a difference.

In our experiments, these regularizations did help in directing the learning towards solutions that generate sparser representations but training with them was problematic mainly for the following reasons. (i) Consistently achieving a level of sparseness is difficult as this does not depend solely on the regularization parameters but is also affected by architecture changes and other hyperparameter changes. (ii) The representations have no imposition to be consistent across different levels of overlap, giving rise to less sparse solutions when there is high overlap. (iii) The level of sparseness between each FBG may not be the same and in fact, the tendency is for the more salient FBG, i.e. the one with higher peak reflectance, to have a narrower representation.

This motivated the construction of a custom regularization. The essence of the required regularization shifts from the sole problem of sparseness and instead focuses more on "narrowness", since making the representation close to a Dirac delta also achieves a sparse representation. The Dirac delta can be derived from a normal distribution with variance tending to zero, inspired by this definition the focus is set on constructing a regularization that shrinks variance. Notice that variance is not referred to as the variance of the neuron outputs, but rather as the variance of a probability density function given by the latent representation.

Following this motivation, the family of central moments is explored, where the first absolute central moment, the second central moment and the fourth central moment are used, which were termed `spread`, `variance` and `kurt`[17]. The nth central moment is defined as $\mu_n = \mathbb{E}[(X - \mathbb{E}[X])^n]$ while the nth absolute central moment is defined as $\beta_n = \mathbb{E}[|X - \mathbb{E}[X]|^n]$. It is chosen to detach [18] the mean since otherwise it would be penalized, and the regularization could move the mean to accommodate the distribution instead of changing the distribution to be close to the mean.

To see the effect these regularizations have, the gradients can be inspected, as these quantify how the input is penalized. In figure 5.4 the spectral shape of an FBG positioned at the center can be seen as input (blue line). As can be seen from the gradients, these are independent of the shape and distribution of input values, instead, they are only dependent on the distance to the mean. This is because each regularization is a distance metric with which the input is weighted and summed, and the gradient of a weighted sum with respect to the input is just the weight vector.

[17]This name is chosen since originally the intend was to use the kurtosis, that is the fourth standardized moment, but it proved to be ill-posed. In consequence, instead the non-standardized version is used in its place

[18]This is a terminology used in `pytorch` that implies that the value of a variable is used as a constant and gradients are not computed through it. For more details look at appendix B.2.

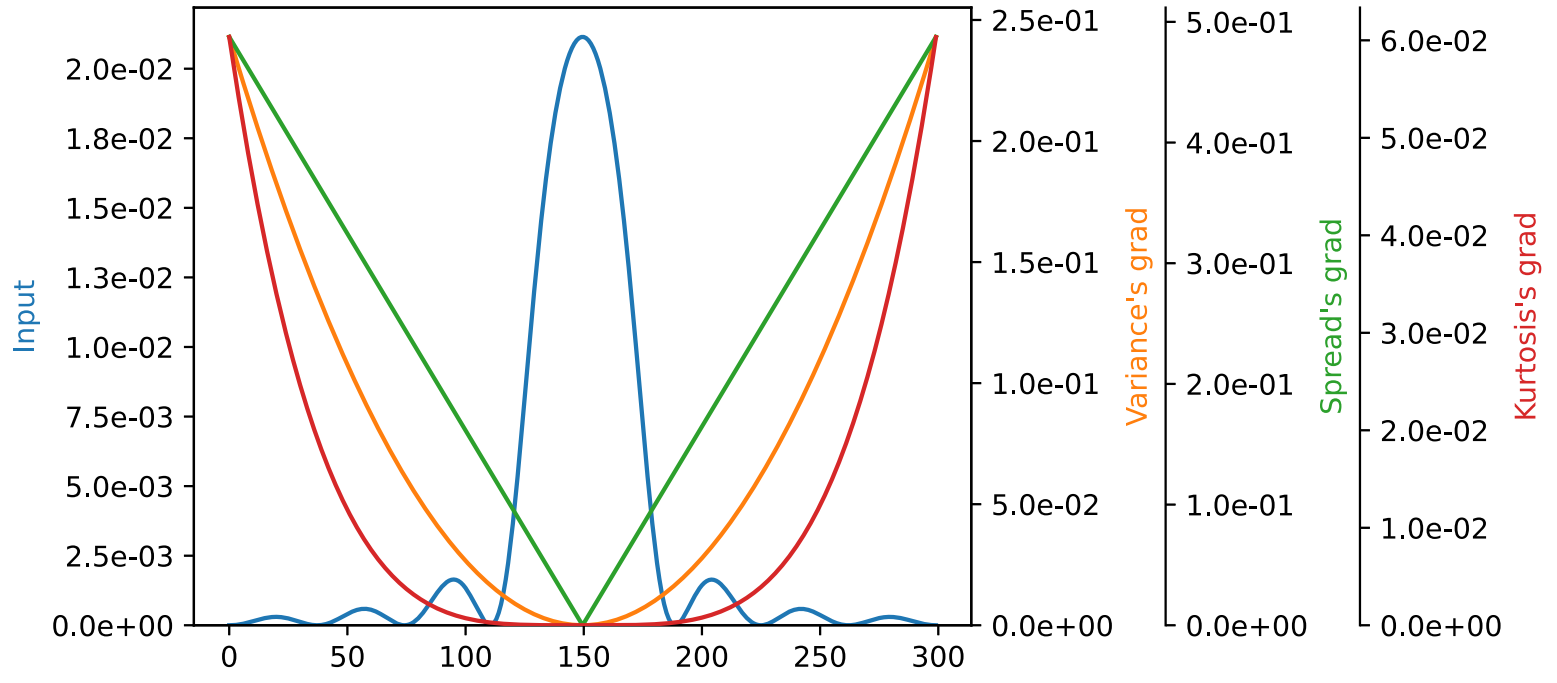


Figure 5.4: Gaussian input (blue) and the gradients with respect to it of the variance (orange), kurtosis (green) and spread (red).

This may seem problematic at first instance since there seems to be no feedback between the input shape and the penalization each element receives. But here it must be considered that the latent variable that will be regularized is the output of a softmax. If instead, the inspected gradients are of an input that when passed through a softmax yields as output the input of figure 5.4, then the gradients with respect to this new input are proportional to the input as can be seen from figure 5.5.

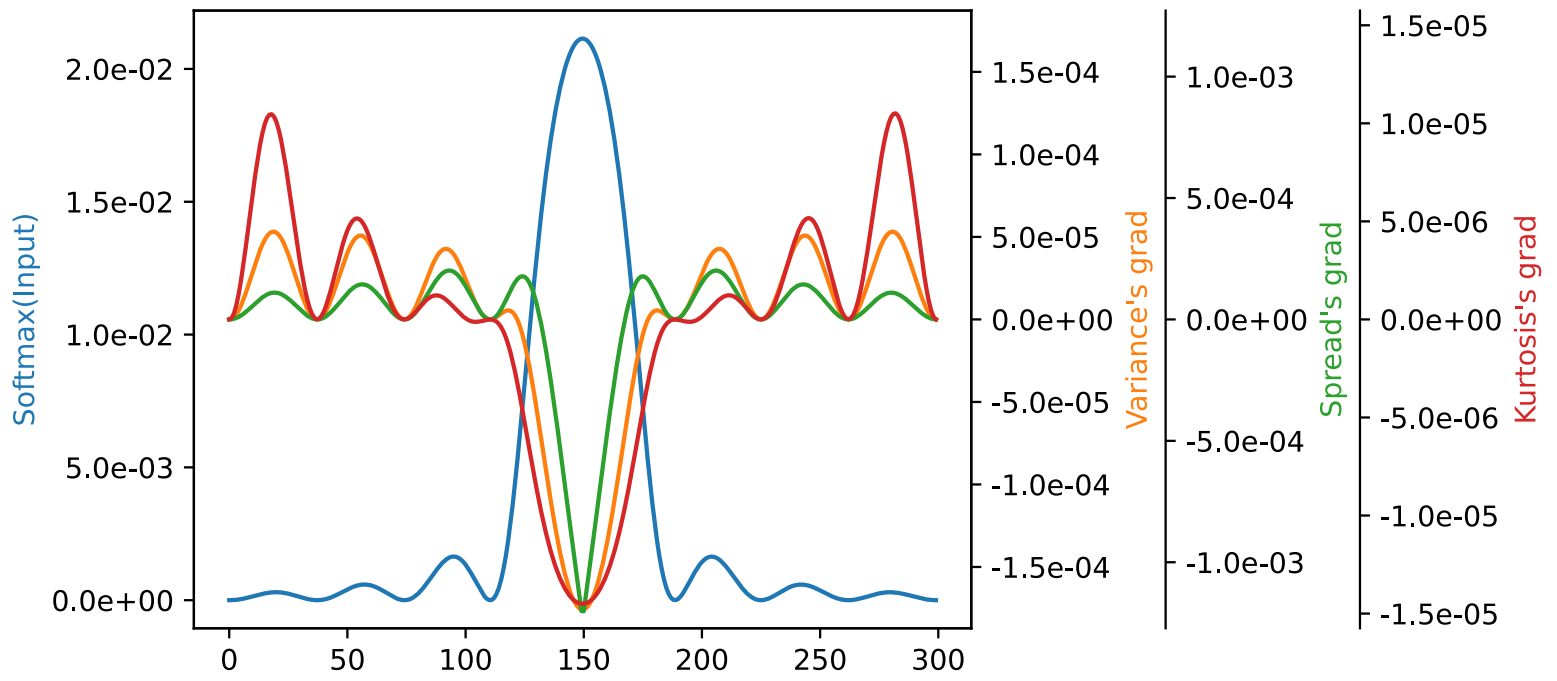


Figure 5.5: Gaussian input (blue) and the gradients with respect to it of the variance (orange), kurtosis (green) and spread (red).

The softmax function is defined in section 4.5, and its jacobian is described by its partial

derivatives given by the following expression

$$\frac{\partial S(x_i)}{\partial x_j} = S(x_i) \begin{cases} (1 - S(x_i)) & \text{if } i = j \\ (-S(x_j)) & \text{if } i \neq j \end{cases} \tag{5.3}$$

where $S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$ and $i, j \in [1, \cdots, N] \times [1, \cdots, N]$. Here the input is the vector x and x_i is the i-th element. The partial derivative $\frac{\partial S(x_i)}{\partial x_j}$ is proportional to the i-th output weighted by its complement to one when $i = j$, while it is weighted by the negative of the j-th output if $i \neq j$.

In the context of the introduced regularization family, the softmax's output is then weighted by a distance metric and summed. Given a regularization from the defined family, $R : \mathbb{R}^N \to \mathbb{R}; x \mapsto R(x) = distance(x) \cdot x$ has a gradient w.r.t the softmax's input given by

$$\frac{dR(S(x))}{dx} = \left[\frac{\partial R(S(x))}{\partial x_1}, \cdots, \frac{\partial R(S(x))}{\partial x_N} \right] \tag{5.4}$$

Each of the partial derivatives is given by

$$\frac{\partial R(S(x))}{\partial x_j} = \frac{\partial (distance(x) \cdot S(x))}{\partial x_j} \tag{5.5}$$

Since the distance metric is detached it can be treated as a constant

$$\begin{aligned} \frac{\partial R(S(x))}{\partial x_j} &= distance(x) \cdot \frac{\partial S(x)}{\partial x_j} \\ \frac{\partial R(S(x))}{\partial x_j} &= distance(x) \cdot \left[\frac{\partial S(x)}{\partial x_1}, \cdots, \frac{\partial S(x)}{\partial x_N} \right] \end{aligned} \tag{5.6}$$

This means that each partial derivative is a weighted sum of the softmax gradient, weighted by the distance metric. From here it is clear that the gradient of the regularization with respect to one of the inputs depends on all the outputs of the softmax. This is why, for example, all the gradients are zero in the center in figure 5.4 but they are negative in figure 5.5.

With this, an adequate regularization family has been constructed but it is still needed to grant the ability to achieve consistent representations across different overlaps and across different FBGs. To achieve this, a weight is added, calculated from the sample standard deviation and tailored to reach a target standard deviation, by making it to be close to one for large differences and approaching zero as it approaches the target. The function that achieves this is defined as

$$f(x) = \frac{relu(std(x) - \sigma)}{std(x)} \tag{5.7}$$

where $relu(x) = max(x, 0)$ and $std(x)$ computes the standard deviation of x. The target standard deviation is an extra hyperparameter termed $\sigma \in \mathbb{R}^+$. Note here, that the standard

deviation must be detached since the function output is intended to be treated as a constant. This function is depicted in figure 5.6 for different targets of σ, where it can be seen that the weight value is one for large values of standard deviation and decreases rapidly when close to the corresponding target value.

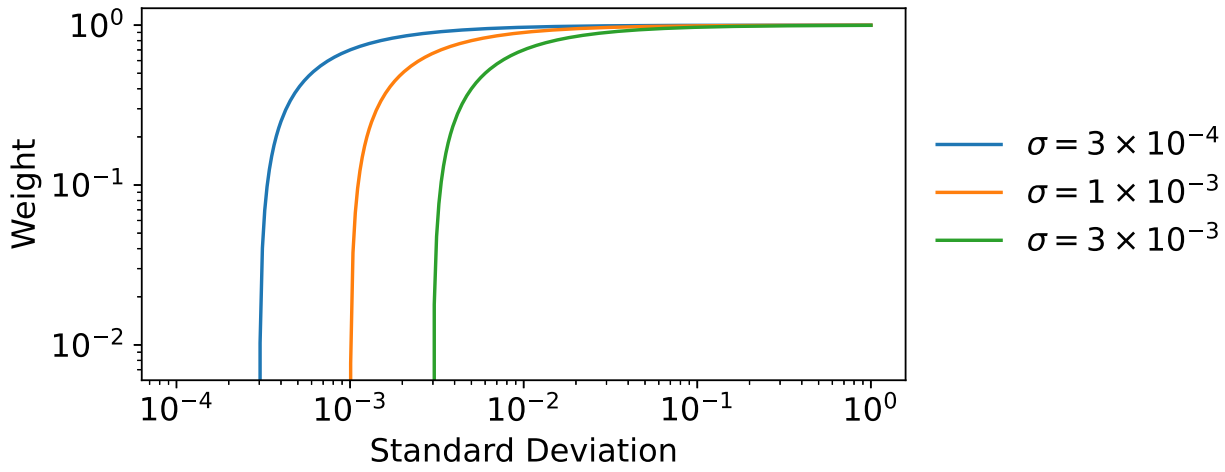


Figure 5.6: Regularization weight function.

Through experiments, it was noticed that the learned latent representation may be noisy if the input contains noise, especially when there is a high overlap. To reduce the noise of the latent representation a roughness penalty is added, as the squared derivative of the normalized derivative. This is a measure of the second derivative magnitude that is scale-invariant, used in the context of smoothing splines [101]. It is given by

$$roughness(x) = \left(diff \left(\frac{diff(x) - mean(diff(x)).detach()}{std(diff(x)).detach()} \right) \right)^2 \tag{5.8}$$

Here $diff(\cdot)$ computes the first-order differences, $mean(\cdot)$ and $std(\cdot)$ the mean and standard deviation, while $.detach()$ represents that the variable is detached and is used to ensure that mean and standard deviation are not penalized.

5.1.4 Decoder

The decoder maps the latent representation to the corresponding spectrum. In the context of CNN-based AE, it is usual to mirror the design of the encoder but using transposed convolution layers[19] instead of plain convolutional layers.

In DL, models tend to be designed to be overcomplete, in the sense that they are arbitrarily complex in relation to the problem they are solving, allowing them to learn the same mapping with different configurations of its parameters. In contrast, the design of the decoder is intended to be as simple as possible, in order to be as close as possible to a complete model. The motivation here is to increase the robustness of the model by easing the update of the mapping. To achieve this as much prior knowledge as possible must be included

[19] This is a DL terminology used to define a convolution that upsamples rather than downsample.

in the model and in this way, design a minimal model that can achieve the required mapping.

From section 2.9 it is known that the process of simulation involves two steps, consisting of the computation of each spectrum followed by the calculation of the joint spectrum. This is illustrated in figure 5.2, where from $\tilde{y}$ the separated spectra vector $\tilde{x}$ is obtained and then from this the joint spectra estimate $\hat{x}$ is calculated. In this work, the serial FBG topology is used. Since the exact simulation is too costly, instead the approximation described in 2.9.1 is used.

The sampling property of the Dirac delta states that by convolving it with a function, it has a translation effect. A discrete equivalent can be constructed from a Kronecker delta followed by a discrete convolution. Since the representation has been constructed to be close to a Dirac delta, this property can be leveraged, and a transposed convolution[20] can be used to generate each spectrum if the kernel corresponds to the spectral shapes.

In practice, it was found that the representation cannot be too narrow since it hinders learning. The trade-off of this is that it resembles a gaussian filter, which is counteracted by the learned kernel, i.e. the learned kernels are noisy. This can be ameliorated by using smoothness regularization over the kernel.

Then the `serial_rec` algorithm 1 to compute the joint spectra is replicated. The implementation is straightforward as `pytorch`'s autograd supports all used operations, it is just needed to add the transmissivity vector as a trainable parameter of the model. As stated in section 5.1.1, the input of the encoder is not normalized. If instead it was normalized, after the last step it would be necessary to normalize the output of the decoder in the same manner. This step would be a complication since it hinders the completeness of the model by allowing multiple configurations of parameters to achieve the same mappings in the decoder.

The model is still overcomplete, since the decoder parameters have no restriction over the range of values they can have. The model can be restricted further by inserting the bounds known its parameters have. The transmissivity vector is bound in the range $[0, 1]$ and the transposed convolution kernels, which represent the spectral shapes, are also bound in the range $[0, 1]$. As seen in section 4.10.2 there are multiple ways of adding parameter constraints to a model, in particular, the approach of parameterization is followed. To achieve a mapping to the desired range a sigmoid function is used. The learning of these reparameterized parameters is improved by the use of natural gradients by modifying the gradient of the sigmoid function to be unity.

Another reparametrization is also used on the transposed convolution kernel, with the objective of evading translation effects over the latent representation, as the spectral shape could be learned in any position of the kernel. One implemented parameterization,

[20] Here a normal convolution could be used but it is explicitly chosen in this way to emphasize that this is a decompressing operation rather than the usual contracting operation associated to convolution layers.

termed *symmetrical*, is achieved by halving the length of the kernel and concatenating it with a flipped version of itself, as depicted in figure 5.7. Using this reparametrization over the convolution kernel comes with the trade-off of hindering the flexibility of the model to represent asymmetrical spectral shapes. Because of this, it is also implemented a reparametrization termed *centered* which consists in representing the kernel as a multiplication of a symmetrical part and a non-symmetrical part. The symmetrical part is constructed as done for the symmetrical reparametrization, while the non-symmetrical part is constructed in a similar manner but with a concatenation by the flipped multiplicative inverse. These two parametrization options are depicted in figure 5.7.

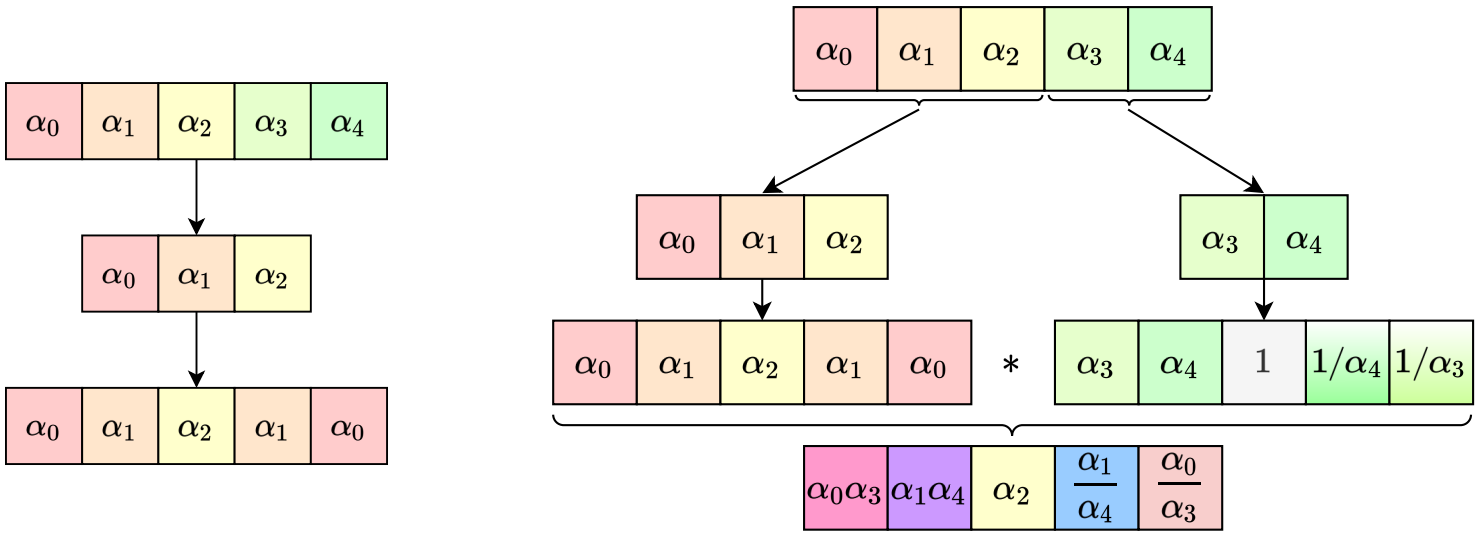


Figure 5.7: Symmetrical parametrization (left). Centered parametrization (right).

5.2 Training Procedure

The architecture space has been described in detail as well as the procedure for generating it. The training procedure is equally determinant of whether an architecture achieves good results. In this section, the design decisions in this respect will be outlined. As has been stated previously, the objective for this architecture is to be able to be trained from simulated data and then be updated from real data in a fully unsupervised manner. These two phases are termed `pre-training` and `finetuning`.

For each training phase, the OneCycle learning rate schedule (detailed in section 4.9.4) is used as it allows convergence in a lower amount of epochs. The learning rate (LR) is chosen according to the LRFinder algorithm described in section 4.9.5. A heuristic called *Gradient Clipping* [102], which consists in re-scaling the gradients above a threshold norm to match that norm, is also used as it allows the use of higher LR values, leading to faster convergence and also adds resilience to high gradient batches that could degrade the performance of the model or even lead to exploding gradients. It was found to be particularly necessary when training with the proposed regularizers that in our experiments could often lead to high-valued gradients that could hinder the convergence of the model.

5.2.1 Pretraining

In the pre-training phase, the model is trained on simulated data. This phase is subdivided into two parts, consisting of the training of the decoder and the training of the encoder. The decoder is trained in a supervised manner by minimizing the regression loss by setting $\gamma = 0$ together with the regularization mediated by the parameters α, σ and ω. In this phase, a suitable member of the dilated encoder architecture family is chosen, with hyperparameters obtained as described in section 5.1.4. The chosen architecture is then trained in an extended manner. To increase the robustness of the model, additive gaussian noise is incorporated to the spectra drawn from $n \sim \mathcal{N}(0, \sigma)$, with standard deviation $\sigma \sim LogUniform(10^{-6}, 10^{-1})$ set per sample, on every batch.

The decoder is not trained, instead, the transposed kernel is initialized with the simulated individual spectral shapes and the transmissivity vector is initialized to the transmissivity values of the simulation.

5.2.2 Finetuning

In the finetuning phase, the model is updated to account for differences between the simulation data and the real data. This change is emulated in the dataset distribution by slightly changing the parameters of the simulation, then the model is finetuned in a fully unsupervised manner by setting $\gamma = 1$ on the loss function. Here the reconstruction error compares the output to $\tilde{x} = x + n$ since, in the real setting, only observations of $\tilde{x}$ would be available and only an approximate x could be obtained by filtering. Furthermore, the noise is sampled from $n \sim \mathcal{N}(0, \sigma = 10^{-2})$ for the whole training set once, and then is fixed for the whole training process.

It was found that training all parameters together yielded good results given that adequate LR values are set for each parameter group. These are the parameters from the encoder, the parameters from the transposed convolution and the transmissivity parameter. To find an adequate LR for each parameter group, the rest of the network is frozen[21] and use the LRFinder algorithm.

5.3 Simulation Validation

In this section, the proposed method will be evaluated and compared with methods from the literature to assess its performance and adaptation capacity. The implemented regressions models are LUT as described in section 3.2.1, ELM and LS-SVR as described in section 3.2.3, and the implemented evolutionary algorithms are GA Binary, GA Real, SDE, DEA and DMS-PSO as described in section 3.2.2. GA Real is a variation of the standard GA algorithm, referred to here as GA Binary, which has continuous representations instead of Binary representations of genes, and only has genes for the spectral positions. The mutation

[21] In the context of DL, this refers to setting a subset of parameters as constants that are not modified.

step is a random fluctuation given by a gaussian distribution centered at zero with variance σ. A swap step is also added to GA Real as done in SDE. For the DMS-PSO algorithm, a swap step is added at the end, as done in [35]; this is described in section 3.2.2. This methods will be referred as *baselines* from now on, while the proposed model trained in a supervised manner is referred as *pretrained model*. The baselines and the pretrained model use a shared simulation, which corresponds to the source domain. In this source domain, the simulation is used either to create the dataset for the case of learning-based methods or to simulate candidates in the case of the EA method. The baselines and the pretrained model are tested in the source domain as a reference. Then they are tested in different target domains, together with the finetuned model, which corresponds to the pretrained model finetuned on target domain data in an unsupervised manner.

The optimal parameters for the regression models ELM and LS-SVR are found through the optimization framework *Optuna* (see appendixB.4) for the case of $Q = 2$. The parameter of the LUT is the number of points in the train dataset. The used parameters are presented in table 5.3. The parameters for the EAs are taken from the references, the population size was increased for some algorithms as the proposed values proved to lead to poor performance. For all EAs a $patience$ parameter was added that stops the search after $patience$-steps of no improvement. The parameters used for the EAs are presented in table 5.4.

Table 5.3: Regression Models parameters.

Model	Parameters
LUT	$M = 10000$ if $Q = 2$ else 30^Q, $distance =$ 'euclidean'
ELM	$hidden\_size = 77444$
LS-SVR	$C = 124349668.96254776$, $\gamma = 0.00817539947388093$, $kernel =$ 'rbf'

Table 5.4: EAs parameters for the two sensor case.

Model	Population Size	Max Generations	Patience	Extra Parameters
GA Binary	20	500	200	$p_{crossover} = 1$, $p_{mutation} = 0.1$, B=10
GA Real	150	500	50	$\sigma = 1[nm]$, $Swap = True$
SDE	30	100	200	$F = 0.8$, $p_{diff} = 0.9$
DEA	200	500	50	$top\_size = 50$
DSM-PSO	3	1000	50	$w = 0.6, pa = 2, ga = 2, lr = 0.1, swarms = 5, migration\_gap = 5$

5.3.1 Unmodified Data

In this section, the performance of all models for the case where the dataset matches the characteristics of the test data are explored. This means that in this setting there is no simulation inaccuracy.

The data used for testing is generated by setting the first sensor (the one with lower reflectivity) to a fixed value of 1550[nm] and sweeping the other FBG from 1549.4[nm]

to 1550.6[nm] over 300 steps. Figure 5.8 shows the distribution of errors of all baseline algorithms for three levels of SNR. It can be noted that most EA algorithms outperform the regressors, except for GA Binary and GA Real for the low noise case. Furthermore, the performance of SDE, DEA and DMS-PSO are roughly on par for all noise levels. It can also be noted that LS-SVR consistently outperforms the other regressors but its performance is worst than the best EA algorithms by orders of magnitude.

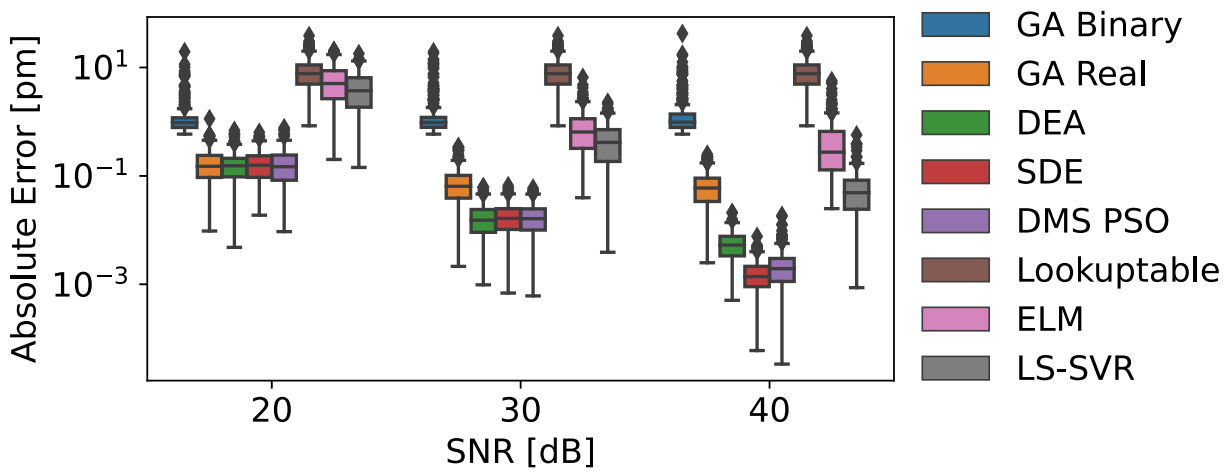


Figure 5.8: Comparison of regressors and EAs

The performance of the pretrained model is inspected in figure 5.9. The left graph displays the true spectral positions (y_1 and y_2) and the corresponding inferences of the pretrained model ($\hat{y}_1$ and $\hat{y}_2$) for each instance of the sweep. The mean absolute error (MAE) is also plotted, corresponding to the per-instance average of the absolute errors between true and inferred spectral positions. On the right graph the distributions of error for each sensor are displayed. The errors are very low, with an average $MAE \approx 3 \cdot 10^{-2}$, which is close to an order of magnitude better than the best baseline regressor LS-SVR. This is evidenced by the almost perfect overlap between true (blue and orange lines) and the inferered spectral positions (dashed green and red lines), as well by the MAE (dashed black line) which displays peaks in the order of 0.1 [pm].

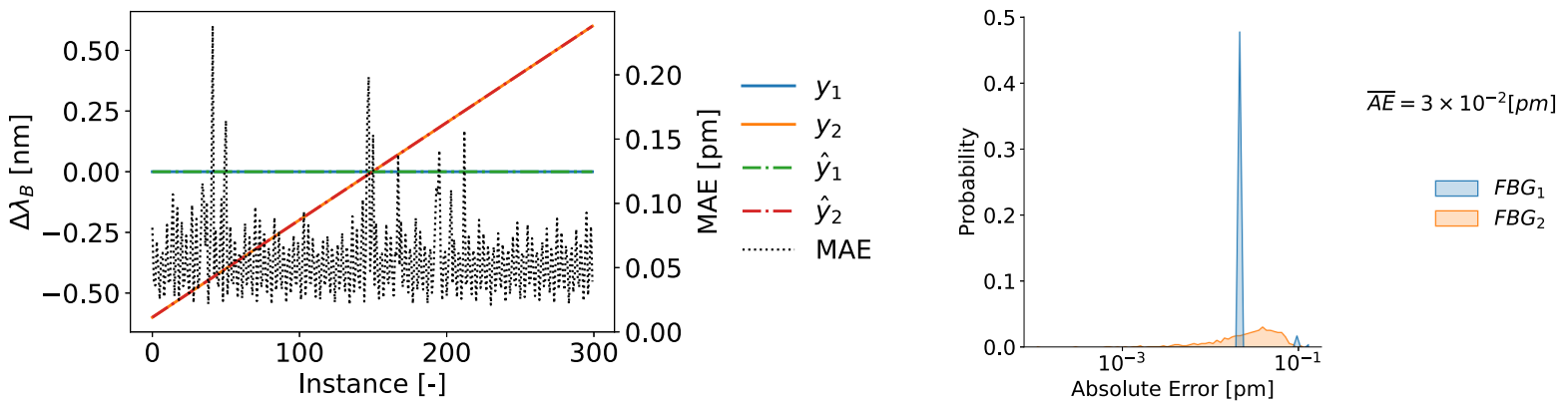


Figure 5.9: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

Now the effect of noise on the inference performance of the model will be inspected. As can be seen from figure 5.10 the performance of the model decreases. The MAE worsens with respect to the noiseless case by roughly an order of magnitude to a value of $MAE \approx 0.2[pm]$ as well as the top error outliers that are in the order of $1[pm]$.

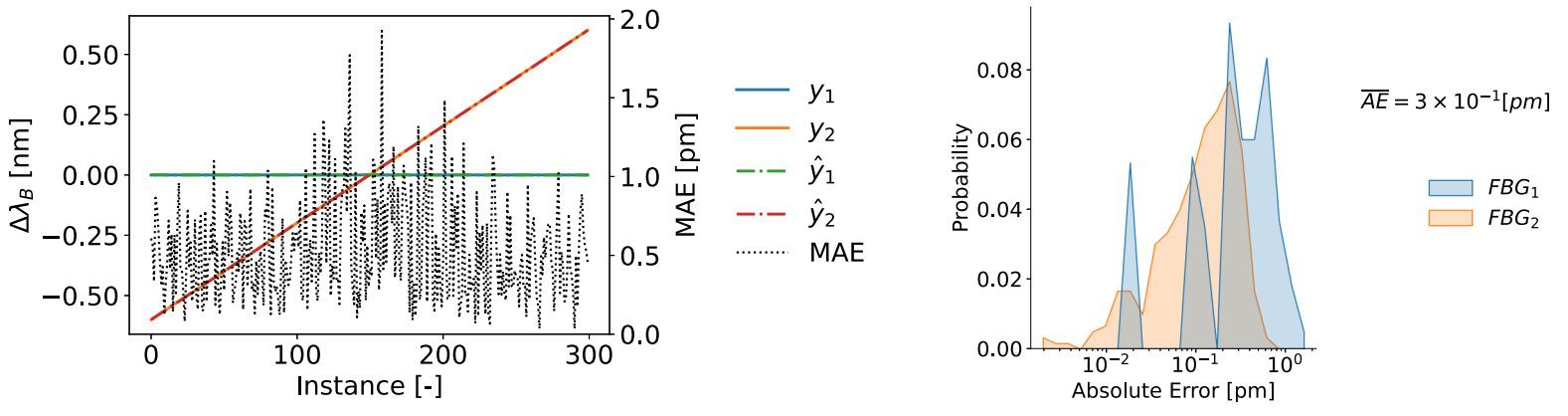


Figure 5.10: Proposed Model Sweep with an SNR of $20dB$ (left). Proposed Model Sweep Error distribution an SNR of $20dB$ (right).

Figure 5.11 shows the error distributions for different levels of SNR, on the left the MAE distributions are displayed while on the right the absolute error distributions are displayed, showing the individual sensor error distributions. It can be seen that for SNR levels above $25dB$ the error distribution remains mostly unchanged and maintains the distribution seen for the case of no noise, displayed previously on figure 5.9. For lower levels of SNR, the performance decreases consistently and decreases more considerably below $10dB$.

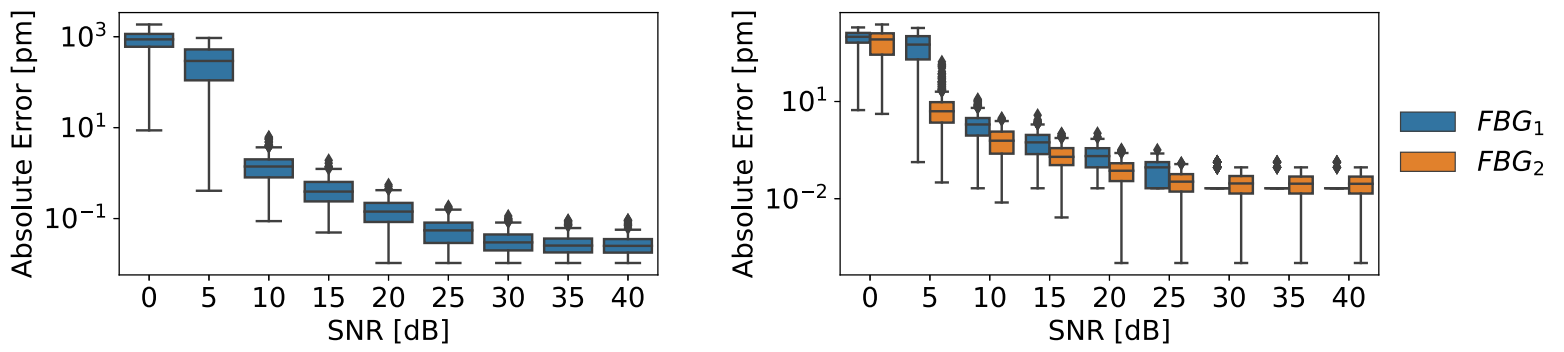


Figure 5.11: Proposed Model aggregated error distributions for different SNR levels (left). Proposed Model error distributions for different SNR levels (right).

In figure 5.12 the aggregated error distributions are compared for the best regressor model and best EA with the proposed model. As can be seen, for low noise the proposed model slightly outperforms LS-SVR but does not outperform SDE. For the high noise case, the proposed model vastly outperforms LS-SVR and closely matches SDE.

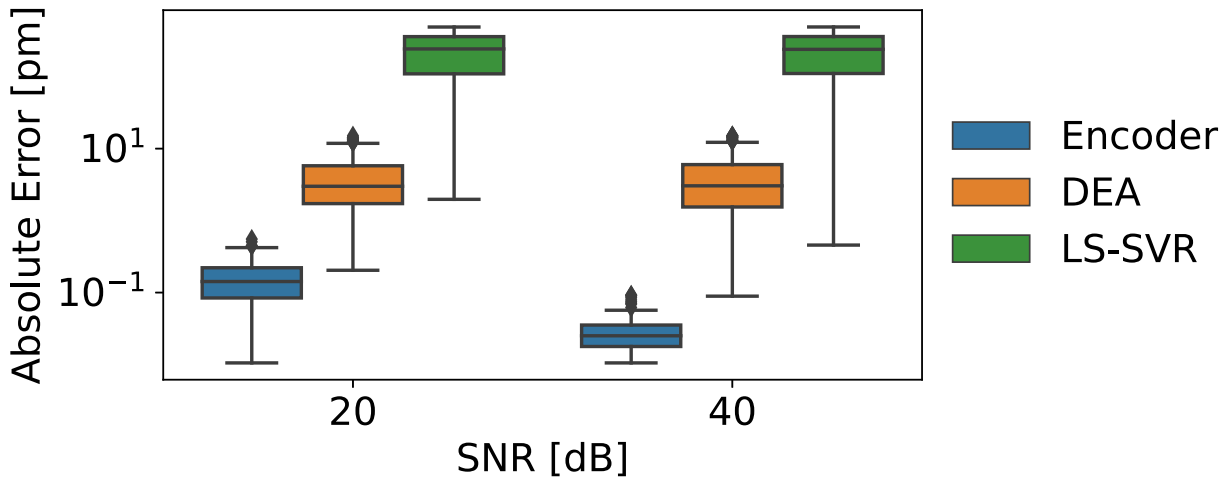


Figure 5.12: Comparison of best regressor and evolutionary algorithm models with respect to the proposed autoencoder model.

5.3.2 Effect of Simulation Inaccuracies

To emulate the differences that would arise when testing on real data, we construct different targets domains, modifying the simulation by adding perturbations to the simulation parameters. These correspond to small changes (about 10%) in the attenuation and profile (shape) of simulated FBGs. Note that here the spectral profile is modified, without affecting the bandwidth, by changing the FBG peak reflectivity value, which, as per the uniform FBG spectral model, affects the spectral profile. This is used as a proxy for simulation inaccuracies that could arise when testing with real data. This should affect the regressor models as the data with which they were trained would be different from the data with which they would be tested. If this is too severe it would suggest that these models should be trained either with real samples or with simulations that correspond very closely to the real data. EAs, on the other hand, rely on simulating candidates to find the best match and as a consequence, it is expected that their performance will be affected by a discrepancy between simulation and real data.

Transmissivity

In this section, the effect of changing the transmissivity parameters of the simulation is addressed. In particular, the transmissivity vector is changed from $A = \{1, 1\}$ to $A = \{1, 0.8\}$. This could represent unaccounted splice losses and the attenuation given by the length of fiber traversed between the sensors.

Figure 5.17 displays the error distributions of all baseline models for different SNR levels. It can be concluded that the introduced simulation inaccuracies significantly impacted the performance of all baseline models. This was more severe for learned regressors suggesting these are not appropriate to train with simulations unless it is very accurate or alternatively should be trained with real data. On the other hand, EA models suffered from a substantial drop in performance but remained constant and in a relatively low error with maximum outliers in the order of $1[pm]$.

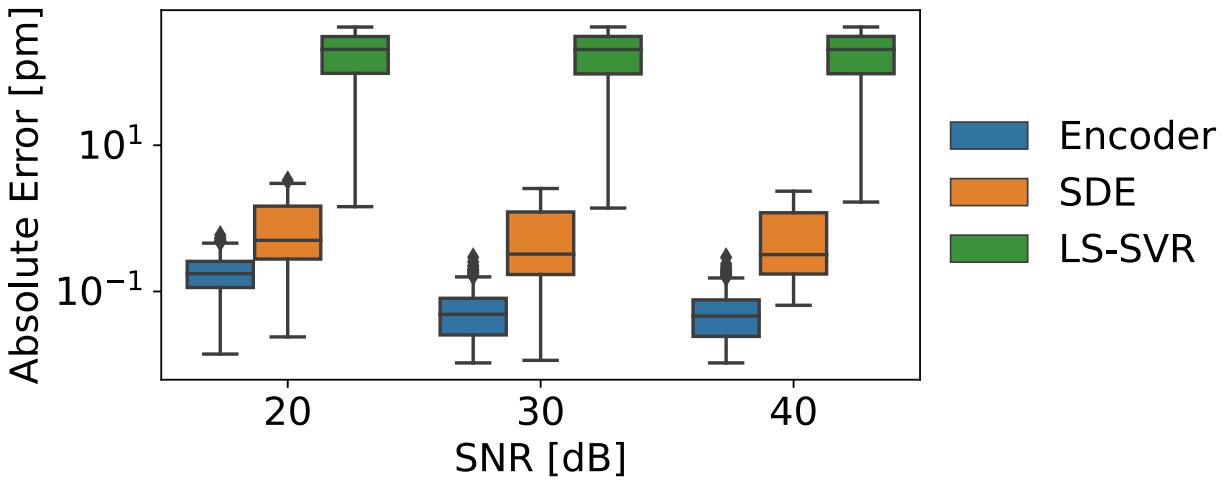


Figure 5.13: Comparison of baseline models

Now the performance of the proposed model will be addressed. As can be seen from figure 5.14 the pattern of error is characterized by having two prominent lobes around the crossing point and smaller lobes further from the crossing point. From the distribution of errors, it can be noted that it has a mean absolute error $MAE \approx 1[pm]$ and top errors in the order of $1.0[pm]$.

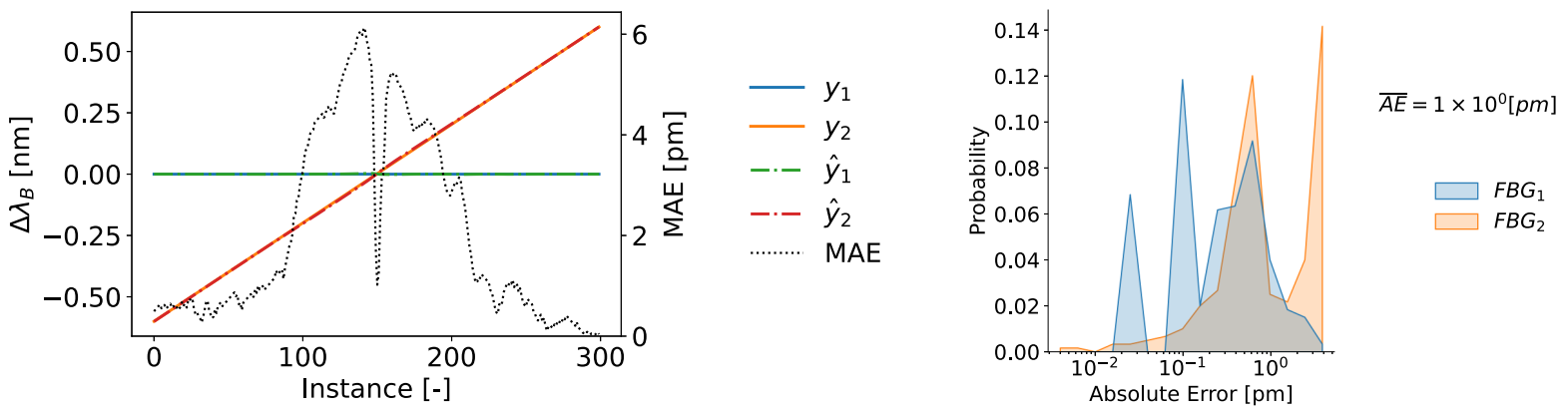


Figure 5.14: Proposed model without finetuning Sweep. Proposed model without finetuning Sweep Error distribution.

The model is finetuned in an unsupervised manner as described in section 5.2.2. The performance increases significantly by more than an order of magnitude, as can be seen from figure 5.15. The mean absolute error is $MAE \approx 6 \cdot 10^{-2}[pm]$ and the top errors are in the order of $0.1[pm]$. It can be observed from the error distribution that the pattern seen previously is lost, although the top errors are still positioned near the crossing point and drop at the crossing point.

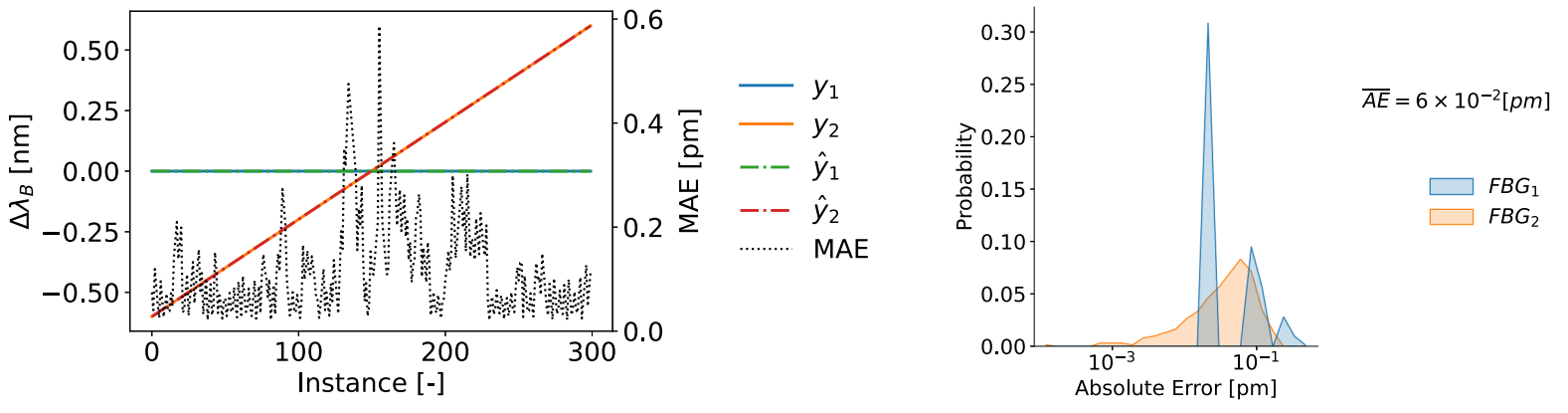


Figure 5.15: Finetuned proposed model Sweep. Finetuned proposed model Sweep Error distribution.

Figure 5.16 shows the performance of the model when the train data matches the characteristics of the test data, and when it does not, before and after the finetuning. As can be appreciated, the finetuning process significantly improved the performance of the model for SNR values above $5dB$, and by more than one order of magnitude for SNR values above $20dB$.

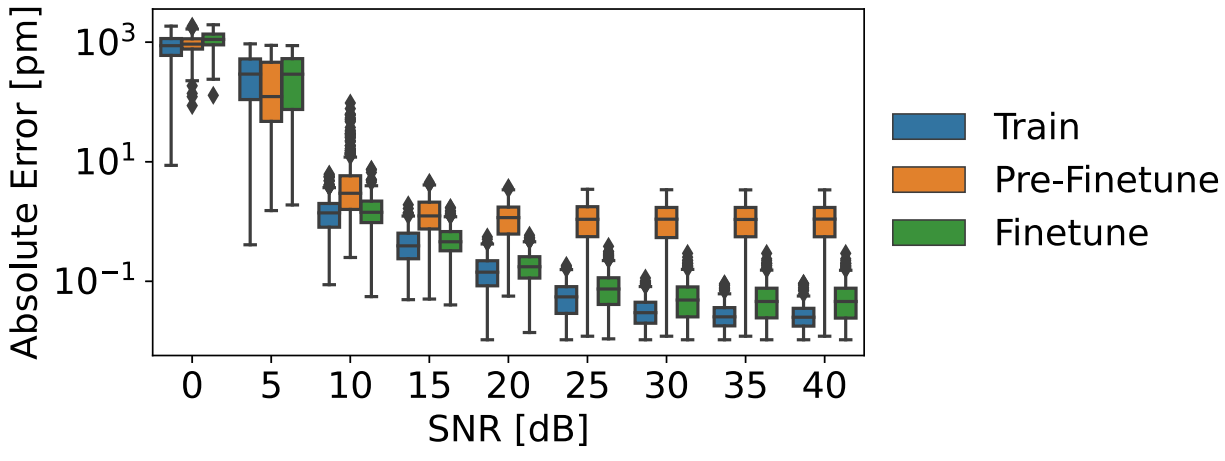


Figure 5.16: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

In figure 5.17 the performance of the finetuned model is compared with the best regressor and AE models. It can be observed that the proposed model consistently achieves subpicometer accuracy, outperforming the other models for all SNR values.

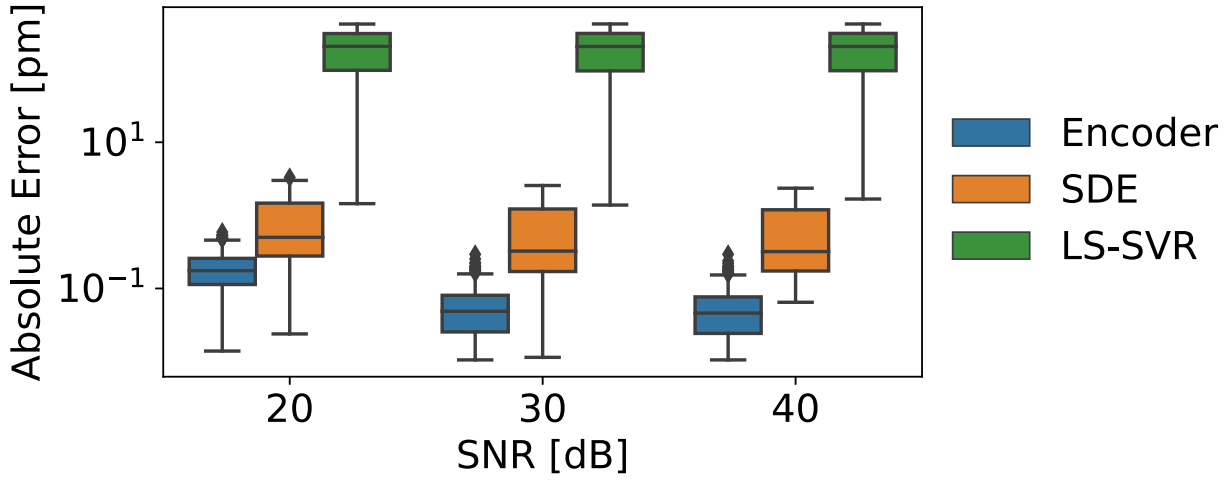


Figure 5.17: Comparison of Finetuned proposed model and the best regressor and AE models.

Peak Reflectance

In this section, the effect of changing the peak reflectance of the simulation is explored. This changes the saturation of each sensor and as a consequence also modifies the spectral shape. In particular the peak reflectance vector is modified from $P = \{0.5, 0.9\}$ to $P = \{0.4, 0.8\}$.

The performance of the baseline models for multiple levels of SNR is displayed in figure 5.22. It can be concluded that spectral inaccuracy in the simulation significantly impacts the performance of all models in a similar manner to the observed in the previous section. The performance decrease is substantially more significant for regressor models. The performance of AE models degrades slightly and is roughly the same for different levels of SNR.

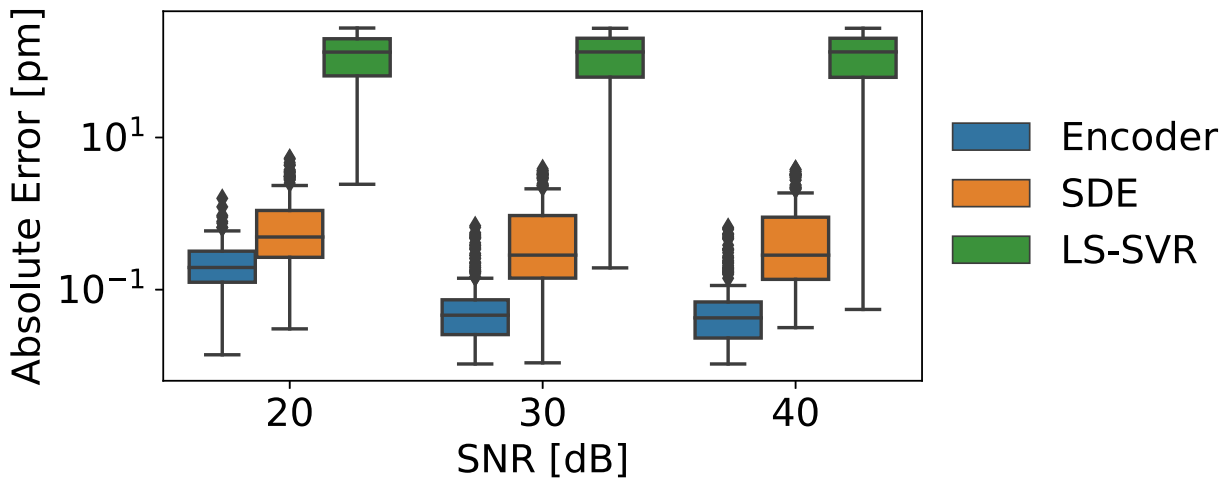


Figure 5.18: Comparison of baseline models.

Now the performance of the proposed model is inspected. As can be seen from figure 5.19, the pattern of error is very similar to the one observed in the previous simulation inaccuracy setting, having two prominent lobes around the crossing point and with smaller lobes positioned further from the crossing point at random positions. The distribution of errors has a mean absolute error $MAE \approx 0.9[pm]$ and top errors in the order of $1.0[pm]$.

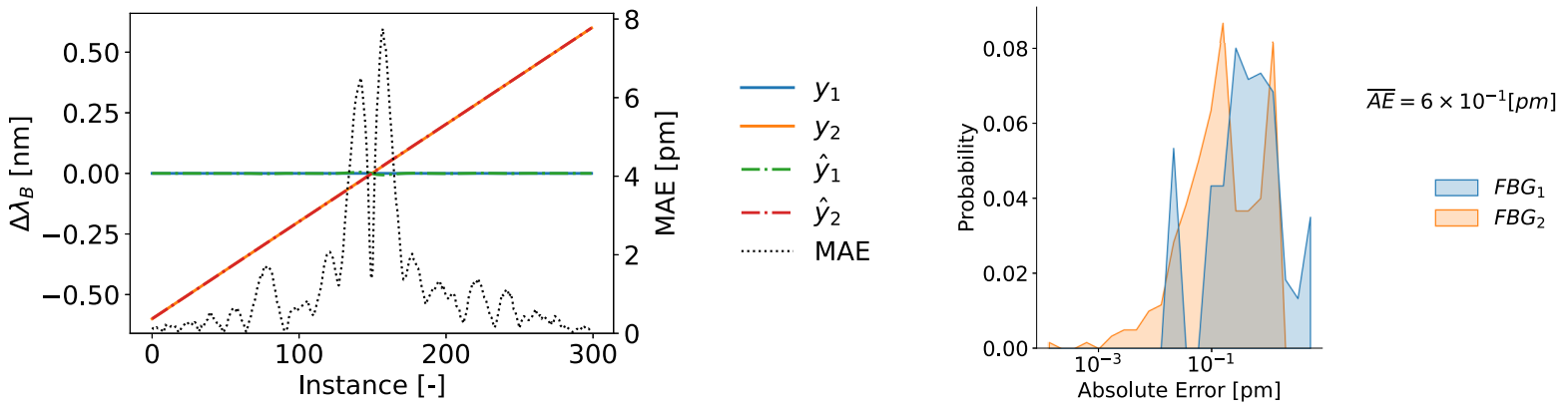


Figure 5.19: Proposed model without finetuning Sweep (left). Proposed model without finetuning Sweep Error distribution (right).

The model is finetuned in an unsupervised manner as described in section 5.2.2. The performance improves significantly by more than one order of magnitude, as can be seen from figure 5.15. The mean absolute error is $MAE \approx 7 \cdot 10^{-2}[pm]$ and the top errors are in the order of $0.1[pm]$. The error pattern is slightly changed with the most distinct feature being two narrow lobes around the crossing point with amplitudes close to $1.0[pm]$.

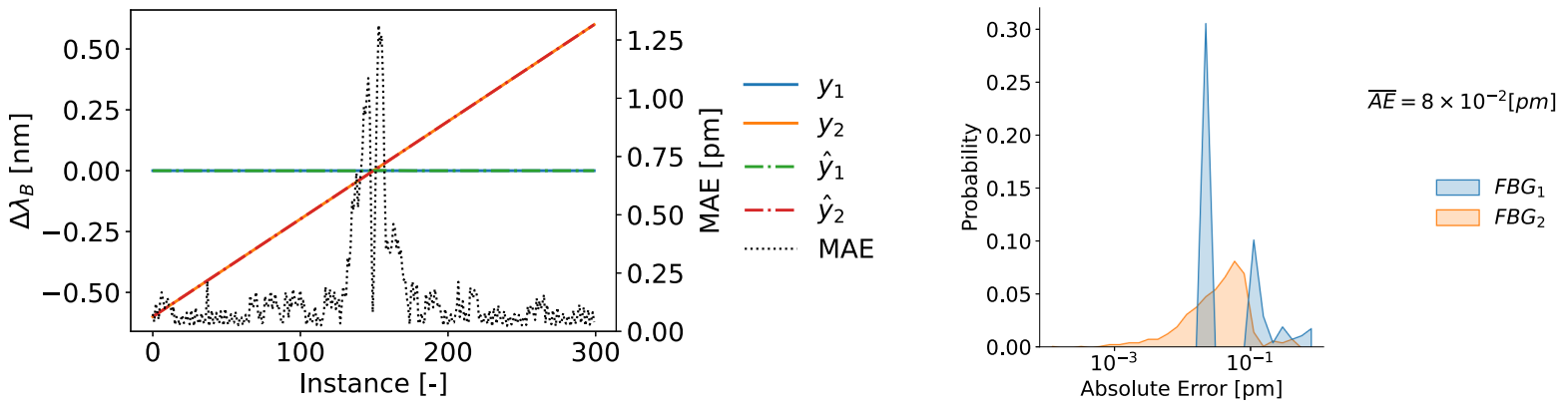


Figure 5.20: Finetuned proposed model Sweep (left). Finetuned proposed model Sweep Error distribution (right).

Figure 5.16 shows the performance of the model when the train data matches test data characteristics, and when it does not, before and after the finetuning. As can be appreciated, the finetuning process significantly improves the performance of the model for SNR values above $5dB$, and by more than one order of magnitude for SNR values above $20dB$.

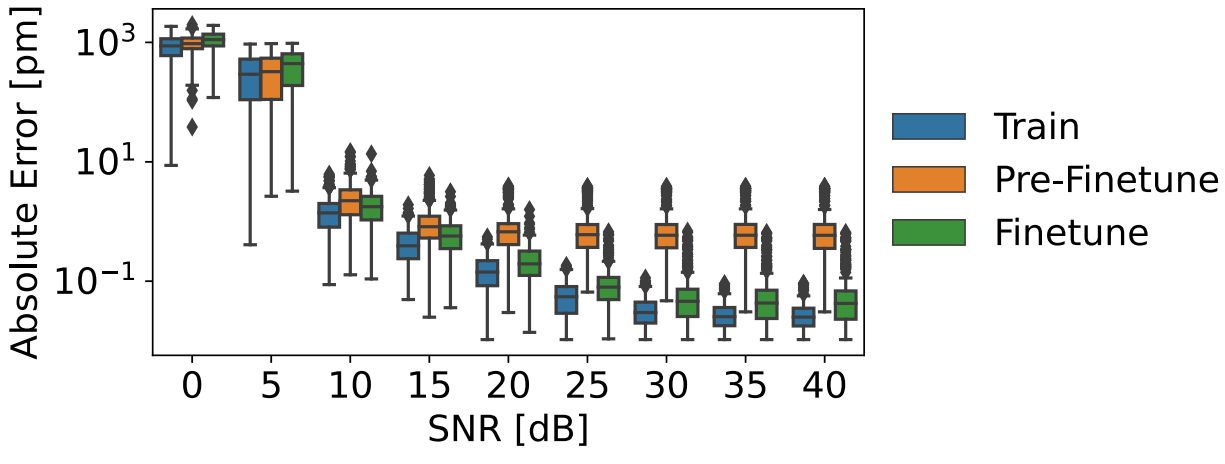


Figure 5.21: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

In figure 5.22 the finetuned model is compared with the best regressor and AE models. It can be seen that it outperforms the other models significantly.

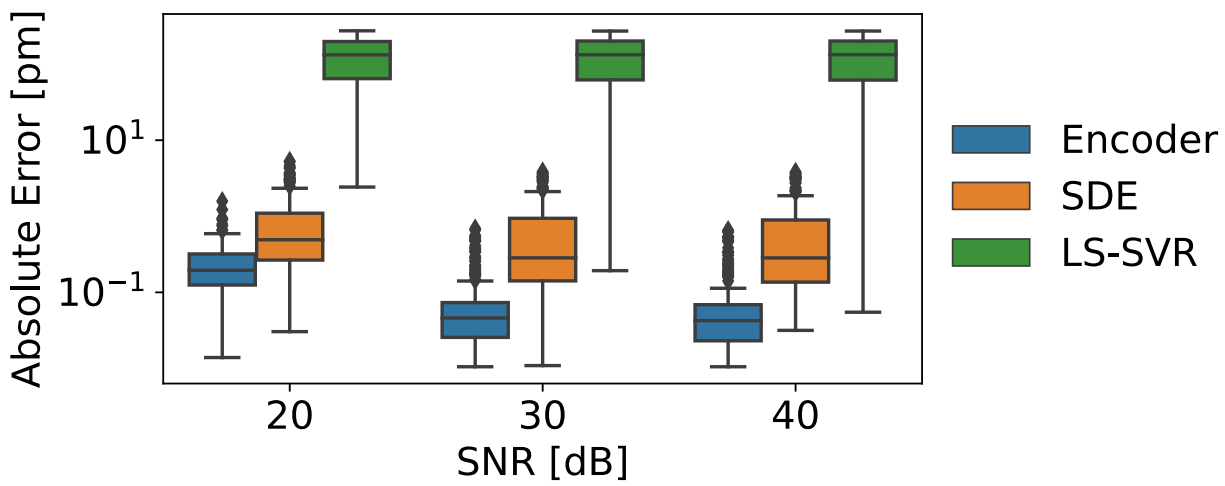


Figure 5.22: Comparison of Finetuned proposed model and the best regressor and AE models.

Effect of Multiple Factors

In this section, the effect of changing simultaneously the transmissivity and peak reflectance values is explored. In particular the transmissivity vector is changed from $A = \{1, 1\}$ to $A = \{0.9, 0.9\}$ and the peak reflectance is changed from $P = \{0.5, 0.9\}$ to $P = \{0.4, 0.8\}$.

The error distributions of all baseline models for different SNR levels are displayed in figure 5.27. It can be concluded that the introduced simulation inaccuracies significantly impact the performance of all baseline models. Similarly to the tendency observed in the previous simulation inaccuracy settings, regressor models substantially degrade their performance. The performance of AE models on the other hand also degraded but to error levels that are not as high and maintained the same performance for all displayed SNR values.

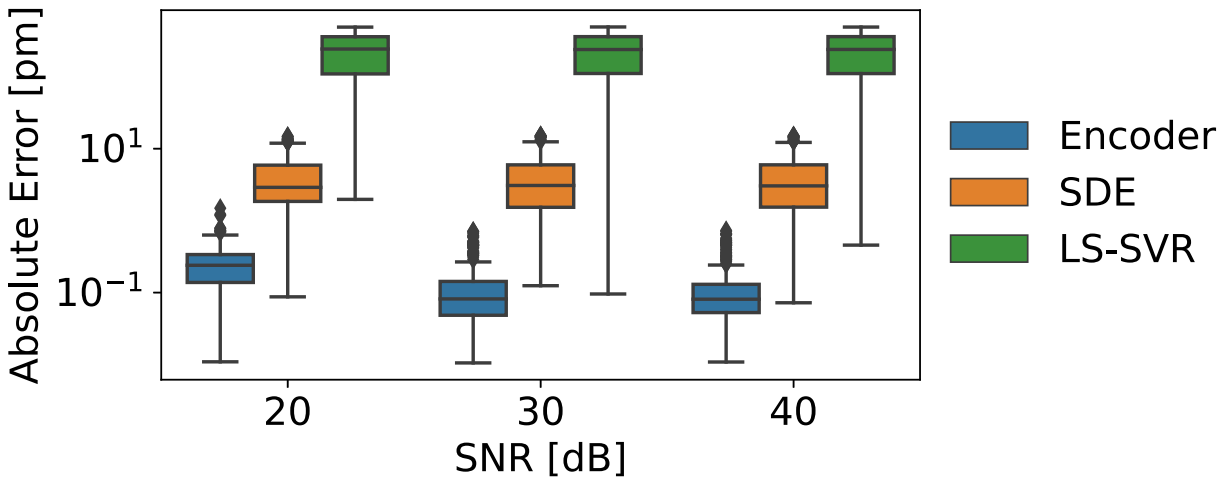


Figure 5.23: Comparison of baseline models.

The performance of the proposed model degrades substantially, more so than what was observed for other simulation discrepancies. The mean absolute error is $MAE \approx 2{\cdot}10^1[pm]$ and the top errors are in the order of $10^2[pm]$ as can be seen from figure 5.24. The error curve maintains the two lobes around the crossing point but also presents other lobes of bigger magnitude further to the sides.

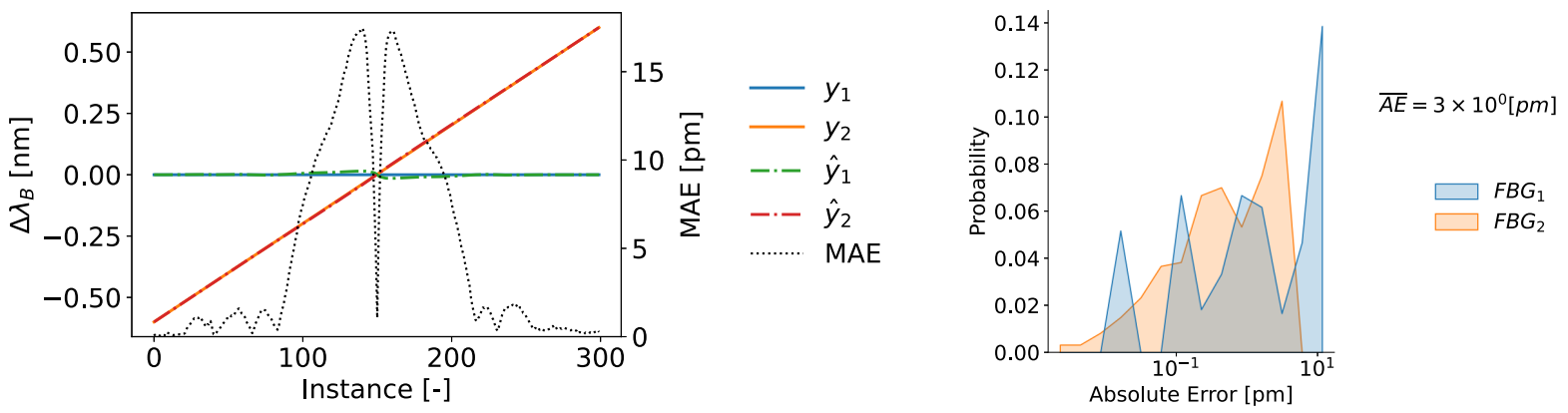


Figure 5.24: Proposed model Sweep (left). Proposed model Sweep Error distribution (right).

The model is finetuned in an unsupervised as described in section 5.2.2. The performance increases substantially by more than one order of magnitude with a mean absolute error of $MAE \approx 0.1[pm]$ and top errors in the order of $1.0[pm]$ as can be seen from figure 5.25. The error curve loses the pattern observed previously and now consists of two narrow lobes around the crossing point with magnitudes in the order of $1.0[pm]$ and lower errors further from the crossing point.

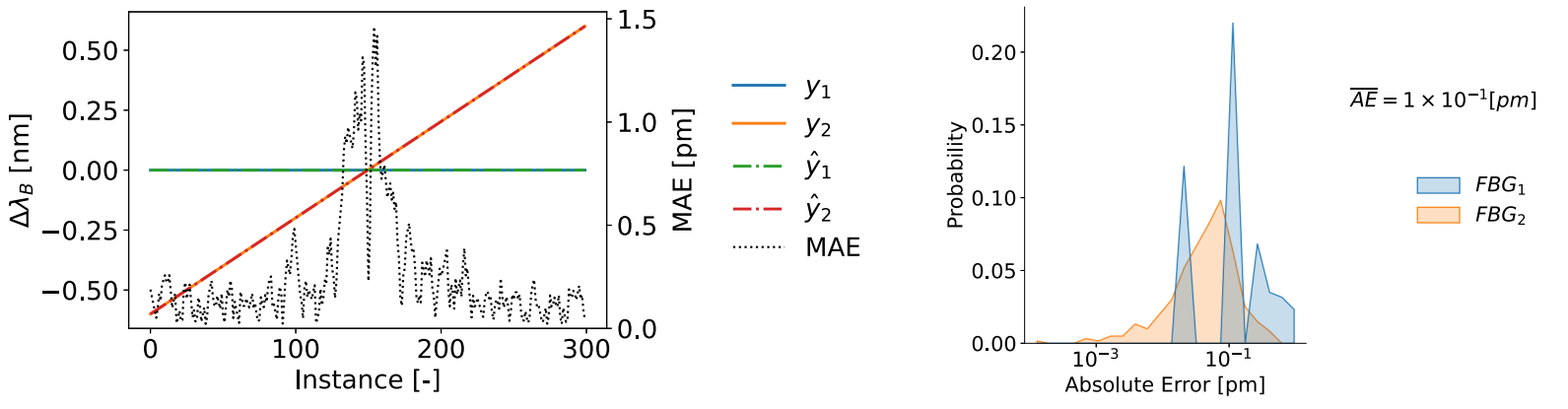


Figure 5.25: Proposed model Sweep (left). Proposed model Sweep Error distribution (right).

Figure 5.26 shows the distributions of error for different levels of SNR for the models when there is no simulation discrepancy and when there is before and after finetuning. As can be seen, finetuning improved the performance substantially for all SNR values above $5dB$, reducing the error by more than one order of magnitude.

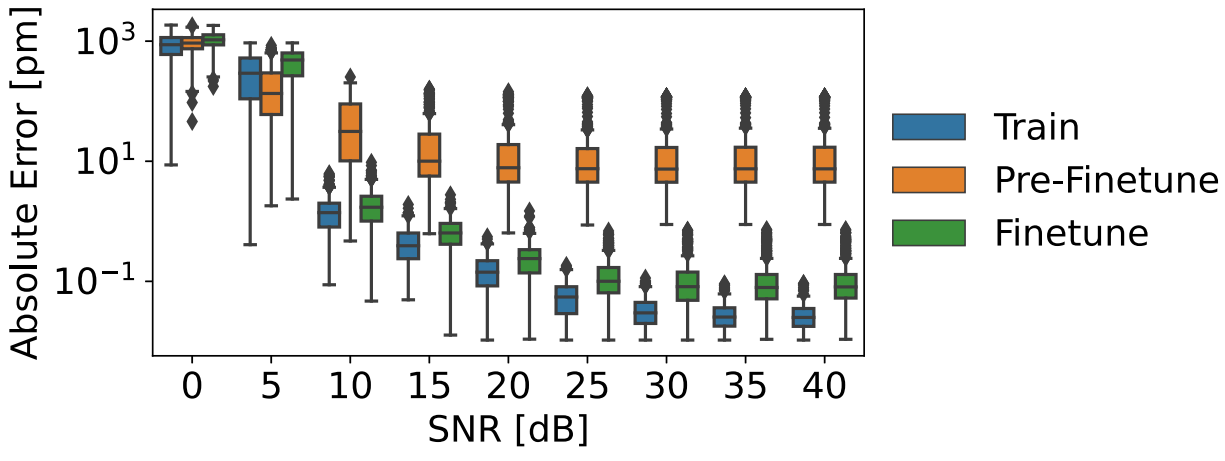


Figure 5.26: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

Figure 5.27 displays a comparison of the finetuned model and the best performing baseline models. It can be observed that it has the lower distribution of error for all SNR values and by significant amounts.

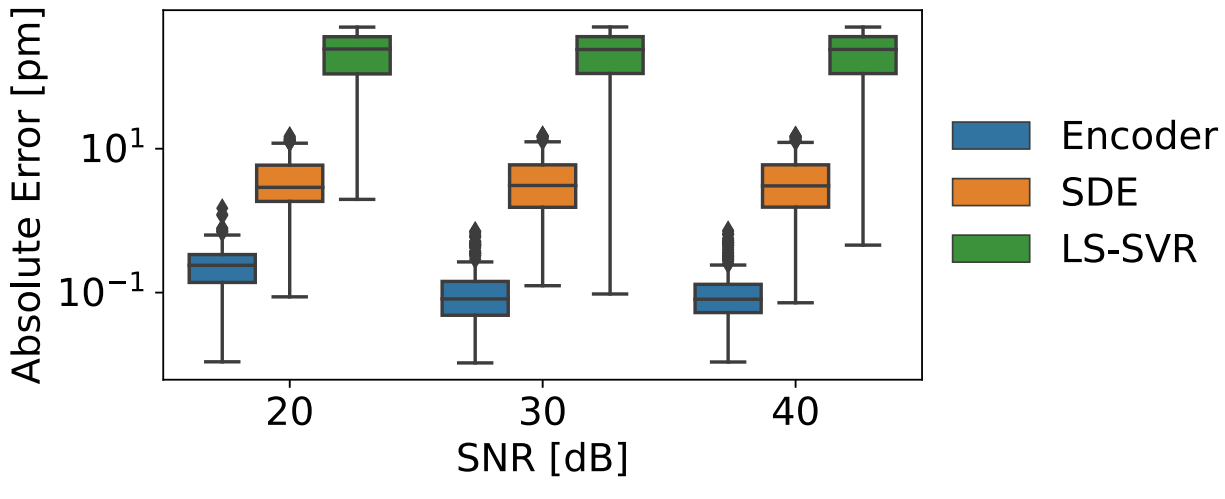


Figure 5.27: Comparison of Finetuned proposed model and the best regressor and AE models.

Figure 5.28 displays the box plots of the MAE distributions of the proposal and the baselines. The *Reference* case corresponds to testing on the source domain and the other cases correspond to the different target domains. In the *Reference* case, the pretrained model outperforms LS-SVR while marginally outperforming SDE. While in the target domains, all baselines have their performance degraded, which is to be expected. On the other hand, the finetuned model returns to an error distribution close to the one obtained by the pretrained model on the source domain, improving the performance significantly.

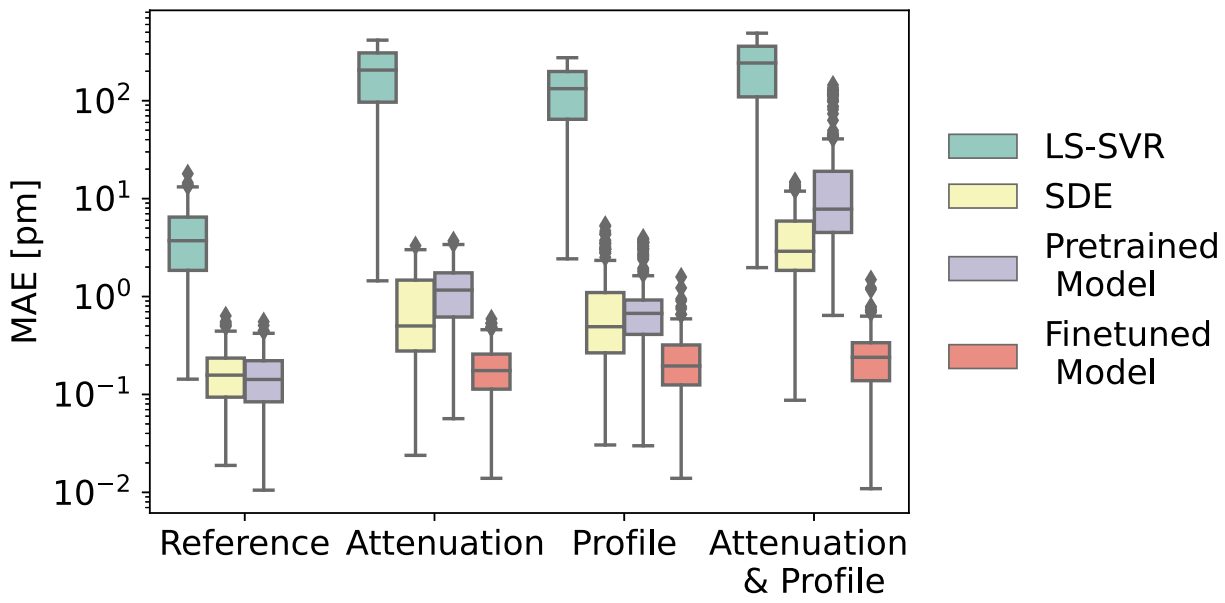


Figure 5.28: Proposal model comparison to baseline models on simulated data for the two-sensor array.

It can be concluded that all baseline models decrease their performance significantly when there are differences between the test data simulation and the train data simulation. In particular, regression models are significantly impacted, consistently achieving errors of several picometers suggesting these models are not suitable for training with simulated data, and should instead be trained with real data. EA models on the other hand experience a performance decrease but in general are bound to acceptable levels, in particular, the last scenario significantly impacted the performance of EA models.

The finetuning procedure over the proposed model proved useful on all simulation inaccuracy scenarios, achieving results similar to the ones obtained with the original model over the test data drawn from the same simulation as the train data. In general achieving consistent subpicometer performance for SNR values above $20dB$, outperforming all baseline models by substantial amounts.

5.3.3 Three Sensors

In this section, the baseline models and the proposed model are benchmarked as done in the previous section, but for the case of one additional sensor. The hyperparameters of the proposed model and the parameters of the regressor models are maintained from the two sensor case. The parameters for the EA models are slightly modified, the detailed used parameters are displayed in table 5.5. The performance of the baseline models is compared to the proposed model as done in previous sections, for different regimes of simulation inaccuracy between the simulation of train data and the simulation of test data.

Table 5.5: Evolutionary Algorithms parameters for three sensors

Model	Population Size	Max Generations	Patience	Extra Parameters
GA Binary	100	500	200	$p_{crossover} = 1$, $p_{mutation} = 0.1$, B=10
GA Real	300	500	50	$\sigma = 1[nm]$, $Swap = True$
SDE	30	100	200	$F = 0.8$, $p_{diff} = 0.9$
DEA	200	500	50	$top\_size = 50$
DSM-PSO	5	1000	50	$w = 0.6, pa = 2, ga = 2, lr = 0.1, swarms = 15, migration\_gap = 5$

Unmodified Data

The baseline models are compared in figure 5.29. It can be concluded that all baselines had their performance affected by the addition of another sensor to the array. In particular, the regressor baselines are significantly impacted, achieving errors well above acceptable subpicometer levels. This is mainly due to the reduction of relative density for each FBG together with the raised difficulty due to closer peak reflectance between sensors, as can be evidenced by the performance observed by the LUT model that deteriorates significantly despite its robust performance observed for the two sensor case. On the other hand, most EA models did not have a significant performance decrease, although higher valued outliers are observed. In particular, DMS-PSO decreases its performance significantly, with outliers several orders of magnitude above its mean value even for high SNR cases. The DEA algorithm achieves good results but has a small number of occasional outliers. The SDE algorithm consistently achieves exceptional results.

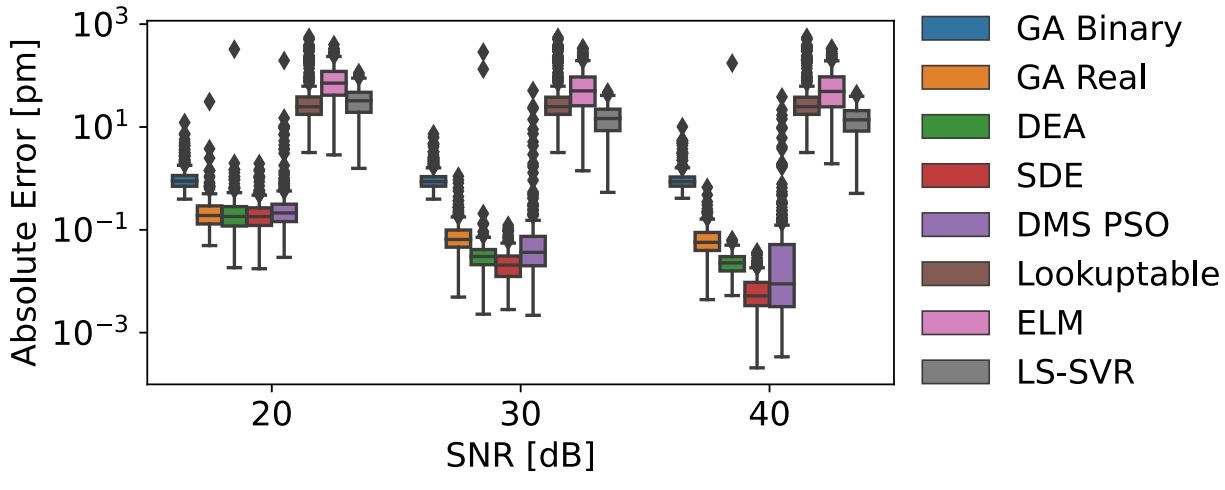


Figure 5.29: Comparison of baseline models

Now the performance of the proposed model is inspected. As can be seen from figure 5.30 the performance deteriorates roughly by an order of magnitude with respect to the two sensor case, with a $MAE \approx 10^{-1}[pm]$ and top error outliers in the order of $1.0[pm]$. From figure 5.30 we can note that the error curve shows the additional difficulty of the inference around the crossing point of the three sensors.

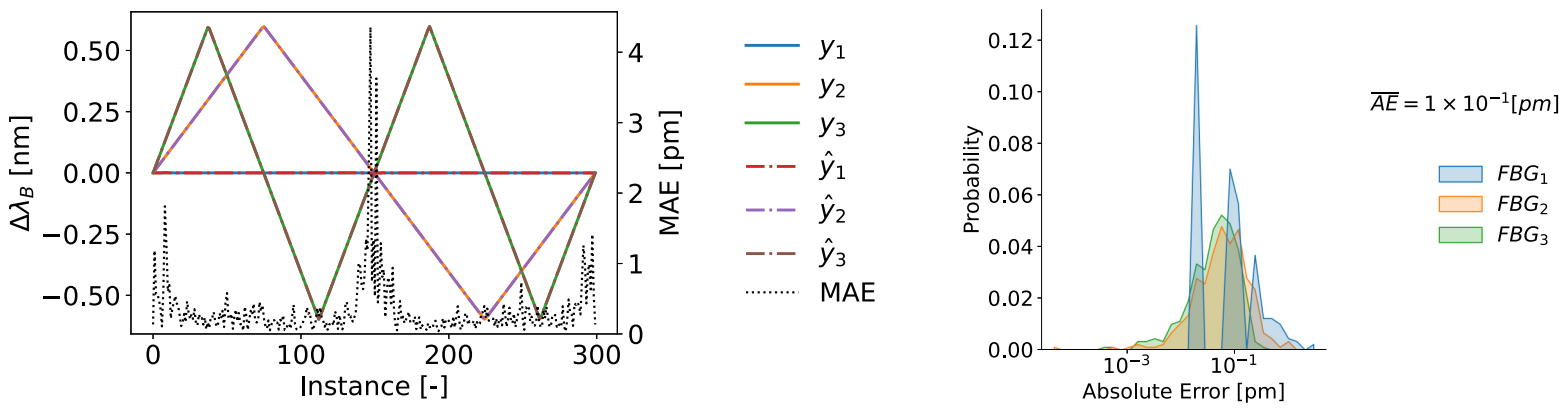


Figure 5.30: Proposed model Sweep (left). Proposed model Sweep Error distribution (right).

It can be observed from figure 5.31 that the error distribution is unaffected for high SNR levels above $25dB$, but it deteriorates gradually for lower values, and more severely below $10dB$. It can be noted that the individual error distributions maintain the tendency of higher errors for the sensors according to their peak reflectance, this is exacerbated further for lower SNR levels.

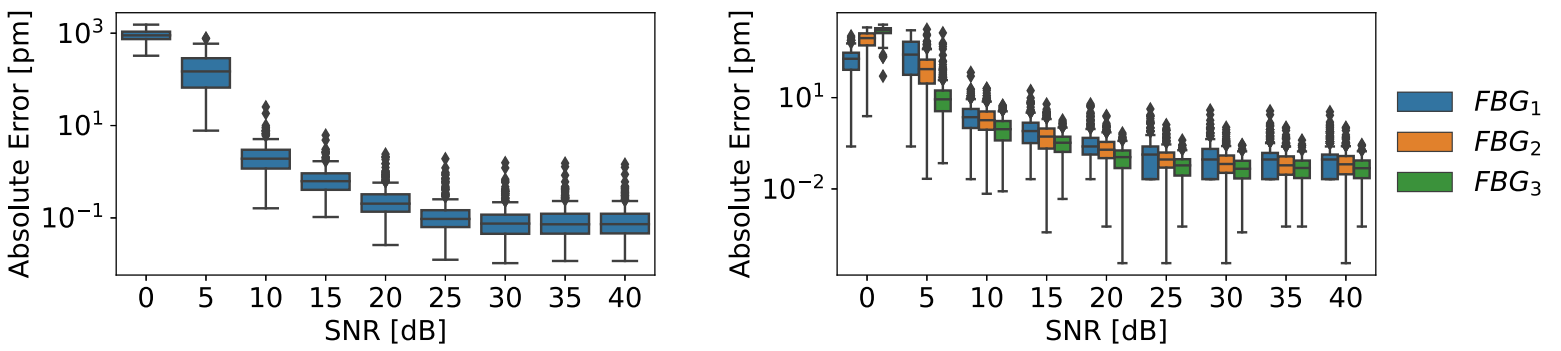


Figure 5.31: Proposed model aggregated error distributions for different SNR levels (left). Proposed model error distributions for different SNR levels (right).

In figure 5.32 the proposed model is compared with the best EA and regression models. It can be noted that the proposed model outperforms by several orders of magnitude the performance of LS-SVR. The model has performance close to SDE for the low SNR case and is outperformed for higher SNR levels.

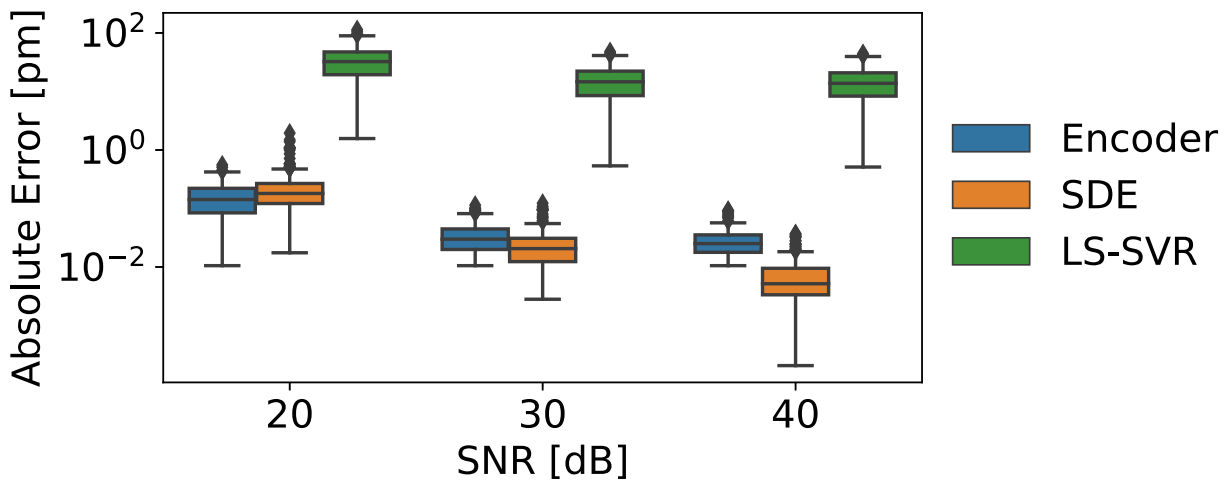


Figure 5.32: Comparison of best regressor and evolutionary algorithm models and proposed model.

Transmissivity

In this section, the case when the simulation inaccuracy is on the transmissivity values will be explored. In particular, the transmissivity vector is modified from $A = \{1, 1, 1\}$ to $A = \{1, 0.9, 0.9\}$.

In figure 5.33 the baseline models are compared for different SNR levels. It can be noted that the performance of all models is significantly degraded. The performance of regression models was already deficient for the unmodified case and is slightly hindered by the simulation inaccuracy. The EA models have better performance than regression models but their performance is degraded substantially from the unmodified case, with MAE values above $1.0[pm]$ and significant high-valued outliers in most AE models.

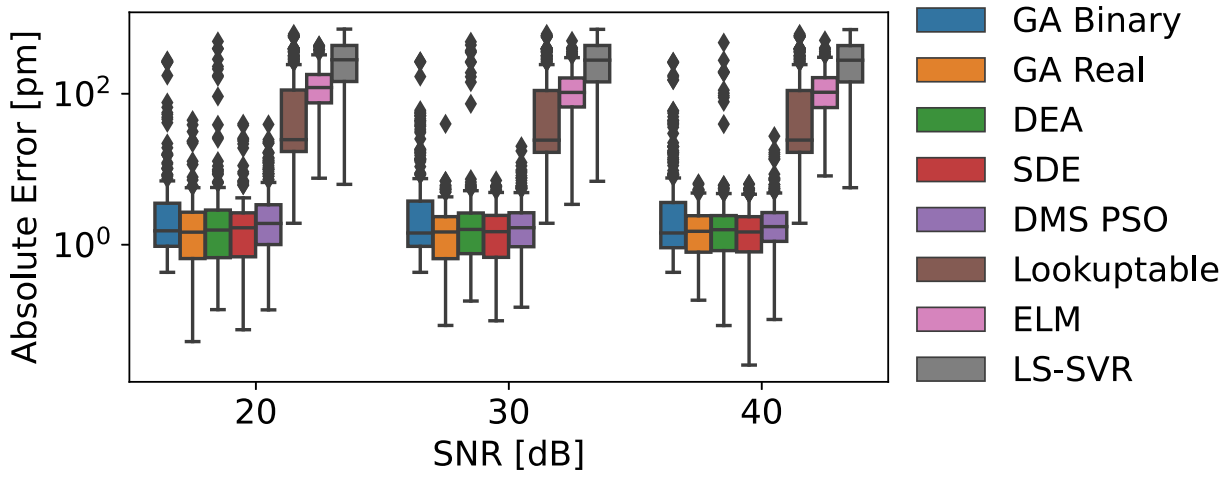


Figure 5.33: Comparison of baseline models

The performance of the proposed model is significantly affected by the transmissivity change as can be seen from figure 5.34 with a $MAE \approx 6.0[pm]$ and top outliers in the order of $10^2[pm]$. It can be observed from figure 5.34 that the error outliers are positioned around the crossing point of the three sensors and correspond to one sensor being inferred in the position of another.

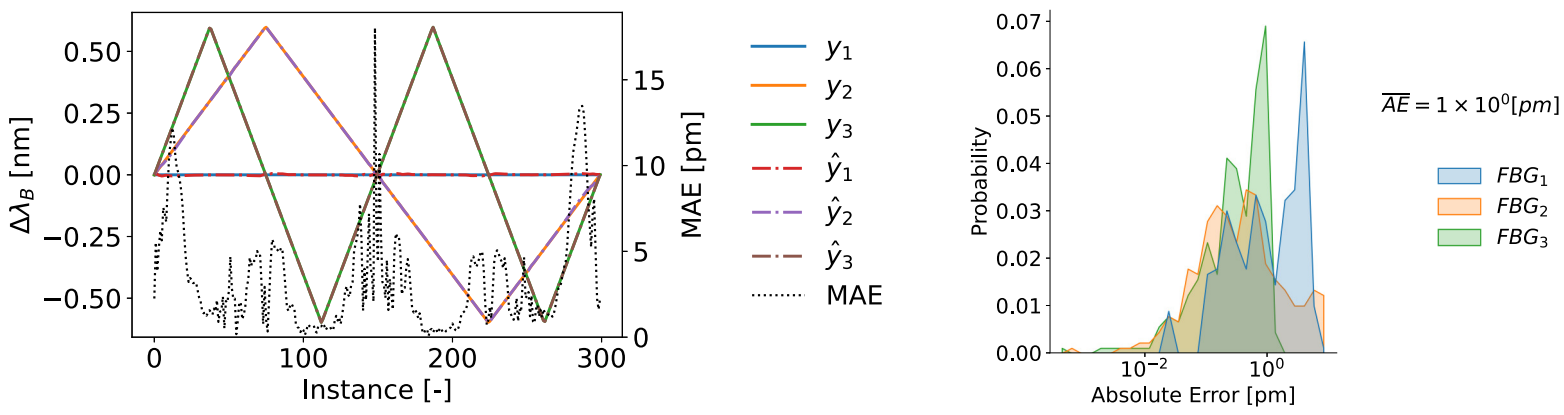


Figure 5.34: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

The proposed model is finetuned as described in section 5.2.2. It can be observed from figure 5.35 that the MAE is significantly reduced by more than one order of magnitude to $MAE \approx 0.2[pm]$ as well as the top error outliers that are now in the order of $1.0[pm]$. It can be noted that the large error outliers are positioned around the crossing point of the three sensors.

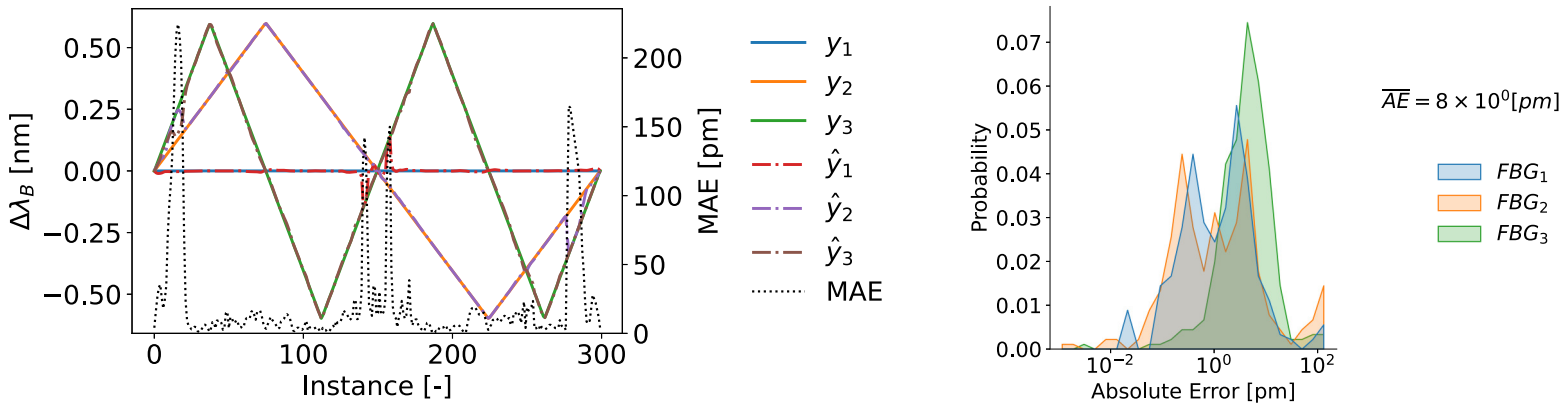


Figure 5.35: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

Figure 5.36 displays a comparison of the error distributions with respect to SNR for the original model tested over the train simulation and the model tested over test data with simulation inaccuracies before and after finetuning. It can be noted that the finetuning process reduced significantly the error distributions, yielding similar results to the ones obtained with no simulation inaccuracy.

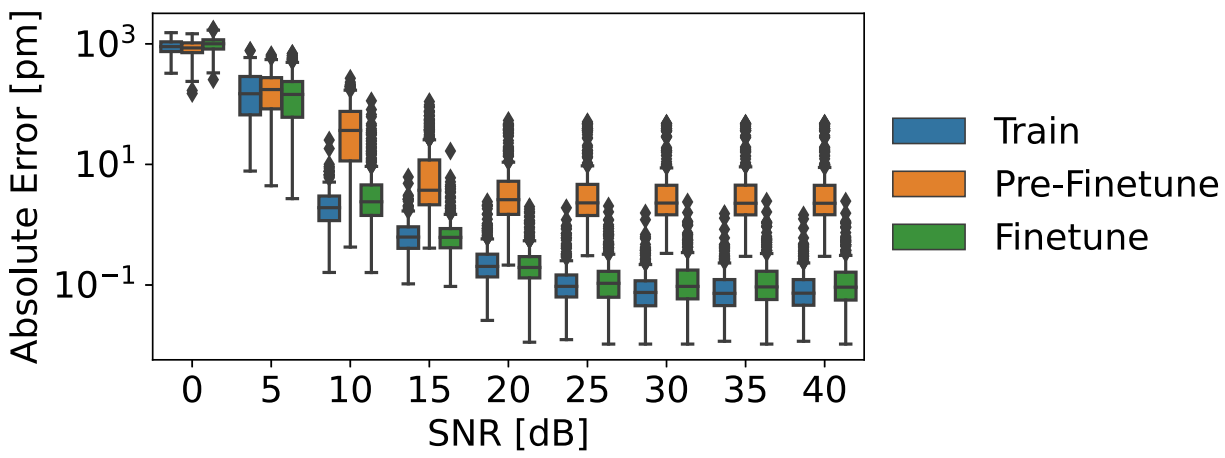


Figure 5.36: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

The finetuned model is compared with the best regression and AE models in figure 5.37. It can be seen that the finetuned model significantly outperforms both models by more than one order of magnitude and has much more compact distributions, especially for the low-noise scenario. It should be noted also that the model has outliers of a couple picometers.

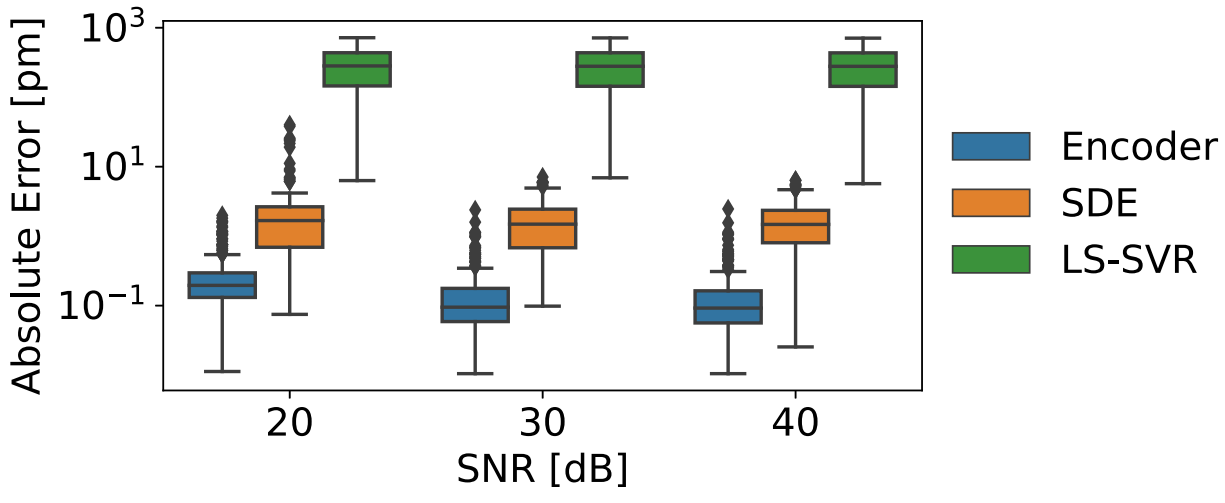


Figure 5.37: Comparison of finetuned proposed model and best baseline models.

Peak Reflectance

In this section, the case when the simulation inaccuracy is over the peak reflectance is explored. In particular, the peak reflectance vector is modified from $P = \{0.5, 0.7, 0.9\}$ to $P = \{0.45, 0.7, 0.95\}$.

The performance for different levels of SNR is compared in figure 5.38. It can be noted that all models are impacted in a similar manner as observed in the previous section. The regression models with already deficient performance in the unmodified case, display slightly deteriorated performance. The AE models have a better performance but are significantly degraded with respect to the unmodified case and with high-valued outliers. It must be noted that GA Binary outperforms all other models, as it finds optimal values for the peak reflectance in conjunction with the spectral positions.

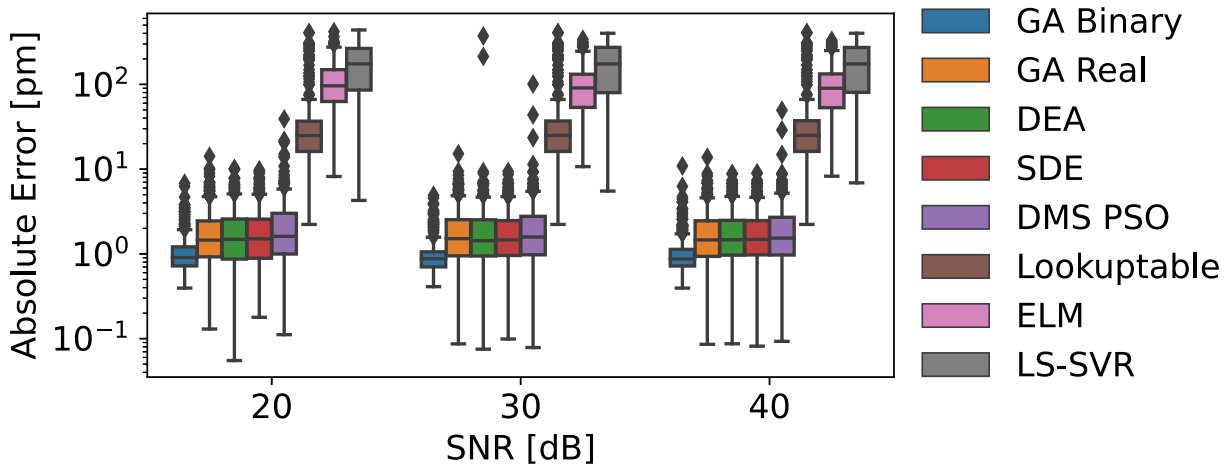


Figure 5.38: Comparison of baseline models.

The proposed model is also affected by the simulation inaccuracy in peak reflectance. This can be observed from figure 5.39 where it can be seen that the $MAE \approx 2.0[pm]$ and the top error outliers are in the order of $20[pm]$. It can be noted that the large errors are positioned around the crossing points, with bigger outliers around the crossing of the three sensors.

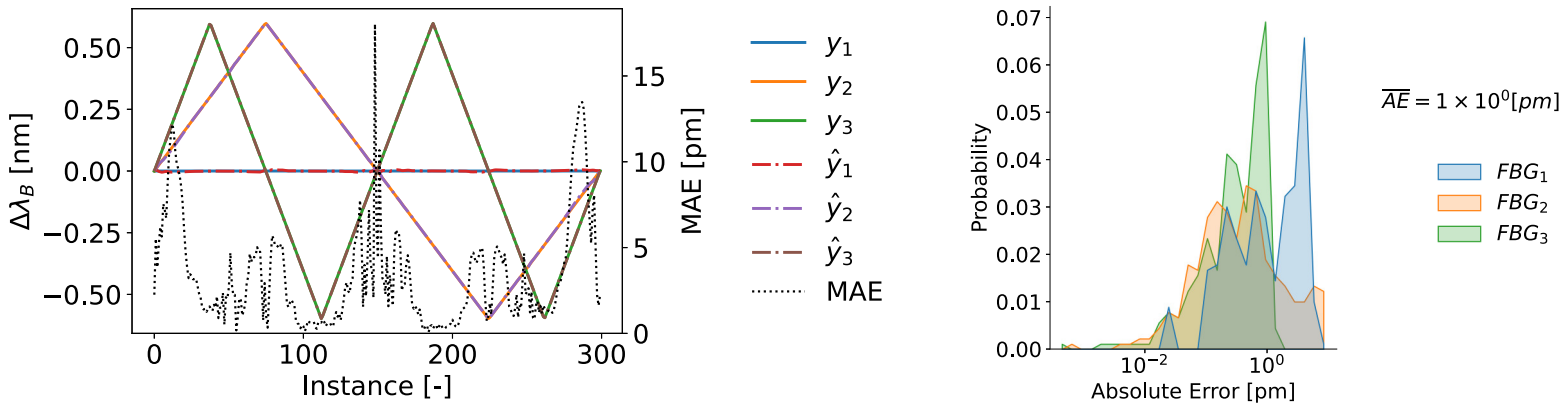


Figure 5.39: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

The model is finetuned as described in section 5.2.2. From figure 5.40 it can be observed that the error distributions are left shifted, now with a $MAE \approx 0.2[pm]$ and top error outliers in the order of $1.0[pm]$. It can also be noted that the top error outliers are positioned around the crossing of the three sensors.

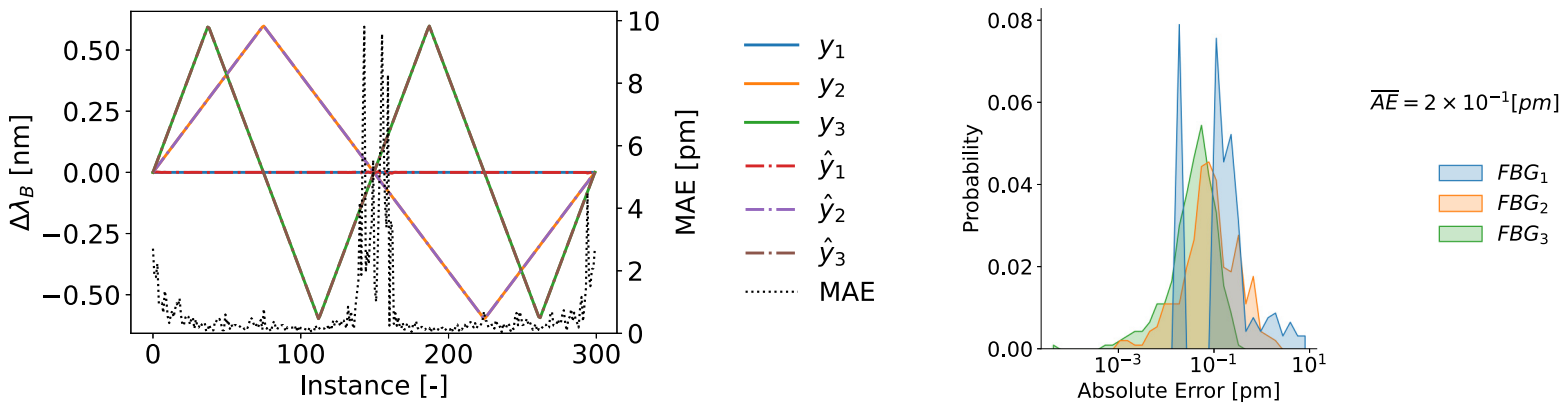


Figure 5.40: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

The performance of the model is displayed in figure 5.41 for the scenario with matching test and train simulation and the scenarios with simulation inaccuracies before and after finetuning. The error distributions are significantly improved, roughly matching the previous performance with no simulation inaccuracy.

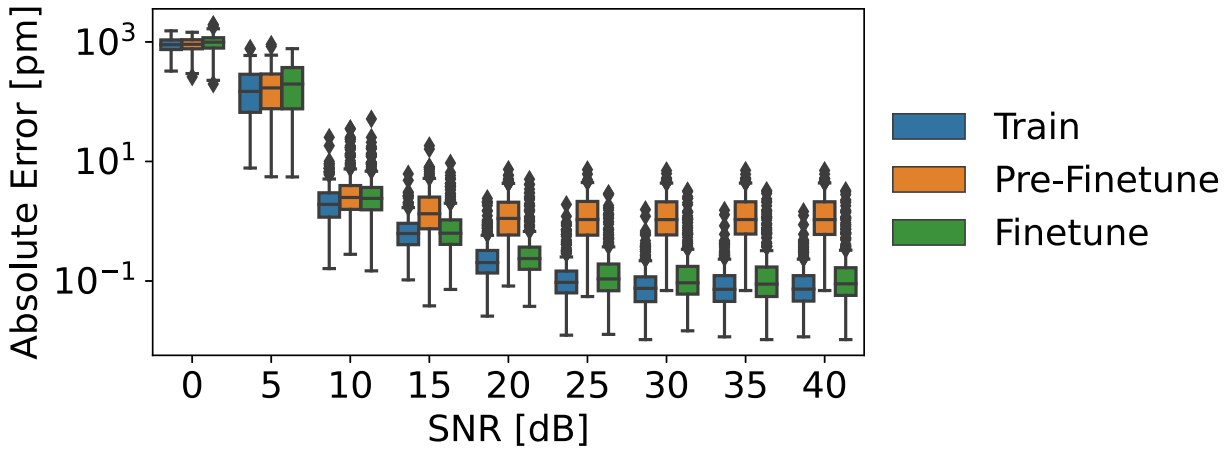


Figure 5.41: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

From figure 5.42 it can be observed that the finetuned model significantly outperforms the best EA and regression models. It can also be noted that the finetuned model has MAE values of more than one order of magnitude lower except for SDE in the low SNR case where it is less than one order of magnitude lower. The model maximum outliers are of a couple picometers.

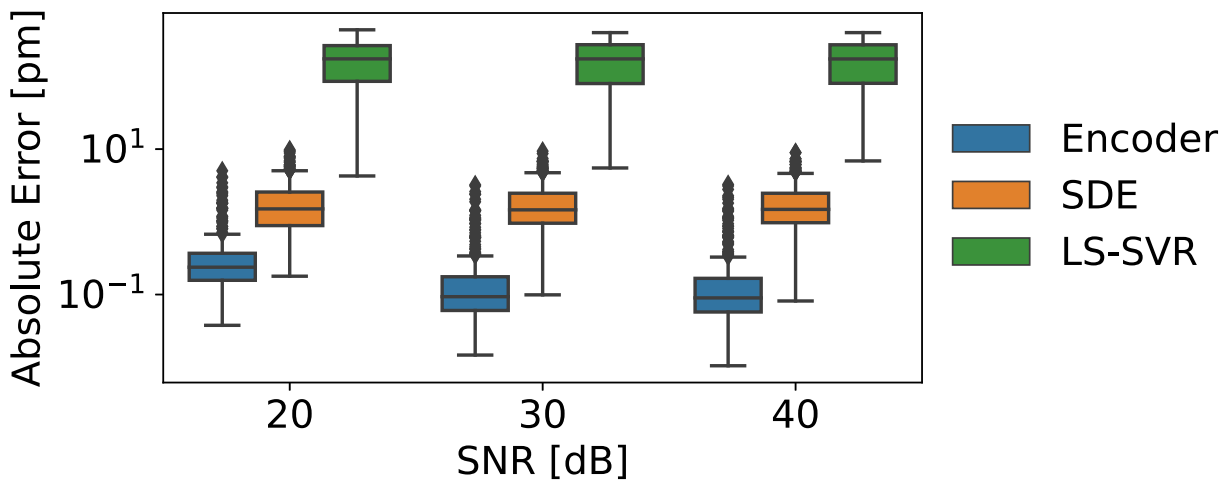


Figure 5.42: Comparison of finetuned proposed model and best baseline models.

Effect of multiple factors

In this section, the case when the simulation inaccuracy is over the peak reflectance and the transmissivity is explored. In particular, the peak reflectance is modified from $P = \{0.5, 0.7, 0.9\}$ to $P = \{0.52, 0.77, 0.99\}$ and the attenuations from $A = \{1, 1, 1\}$ to $A = \{0.95, 0.95, 0.95\}$.

The performance of baseline models for different SNR levels is displayed in figure 5.43. It can be seen that the performance is degraded significantly for all models, with slight differences for the regression models that already displayed poor performance and more severely for AE models. AE models still outperform regression models but display large valued outliers.

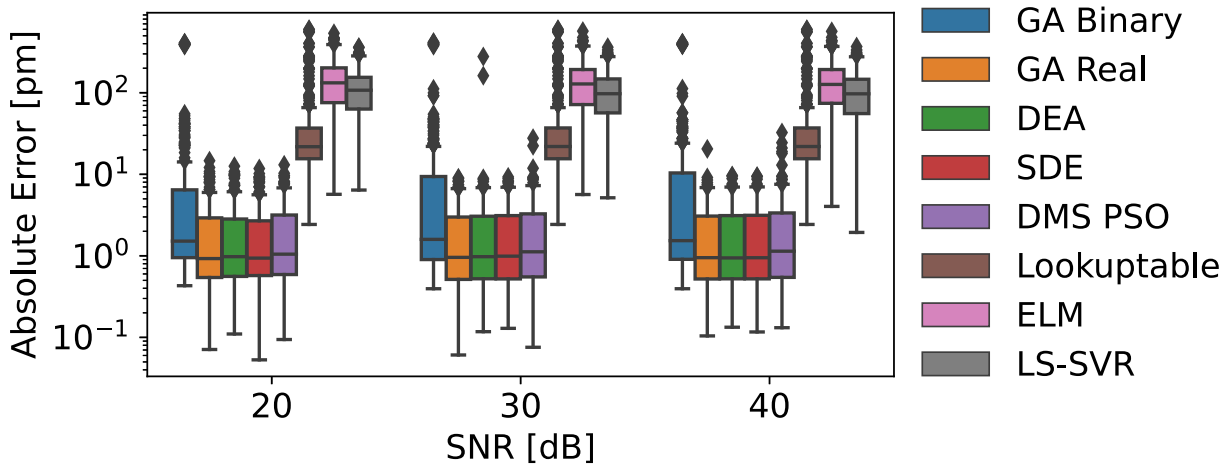


Figure 5.43: Comparison of baseline models.

The performance of the proposed model is degraded by the simulation inaccuracies. From figure 5.44 it can be observed that the $MAE \approx 3.0[pm]$ and the top errors are in the order of $10^2[pm]$. It can also be noted that the error outliers are positioned around the crossing points, with the larger errors around the crossing of the three sensors.

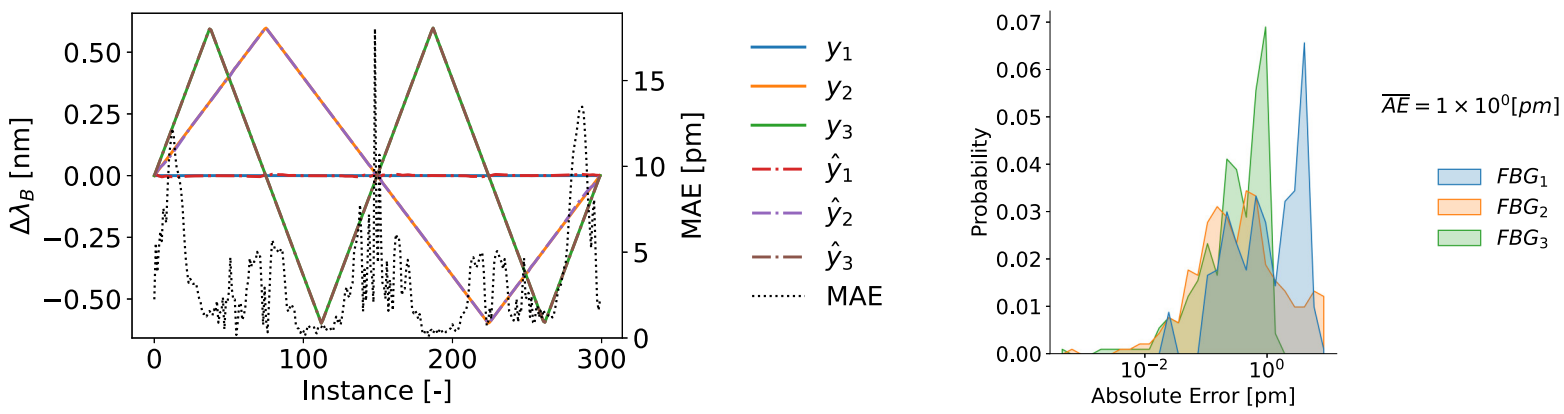


Figure 5.44: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

The model is finetuned as described in section 5.2.2. The errors are significantly lowered as can be seen from figure 5.45, with a $MAE \approx 0.3[pm]$ and top error around $1.0[pm]$. It can be noted that the higher errors are positioned around the crossing of the three sensors with top values around $10[pm]$.

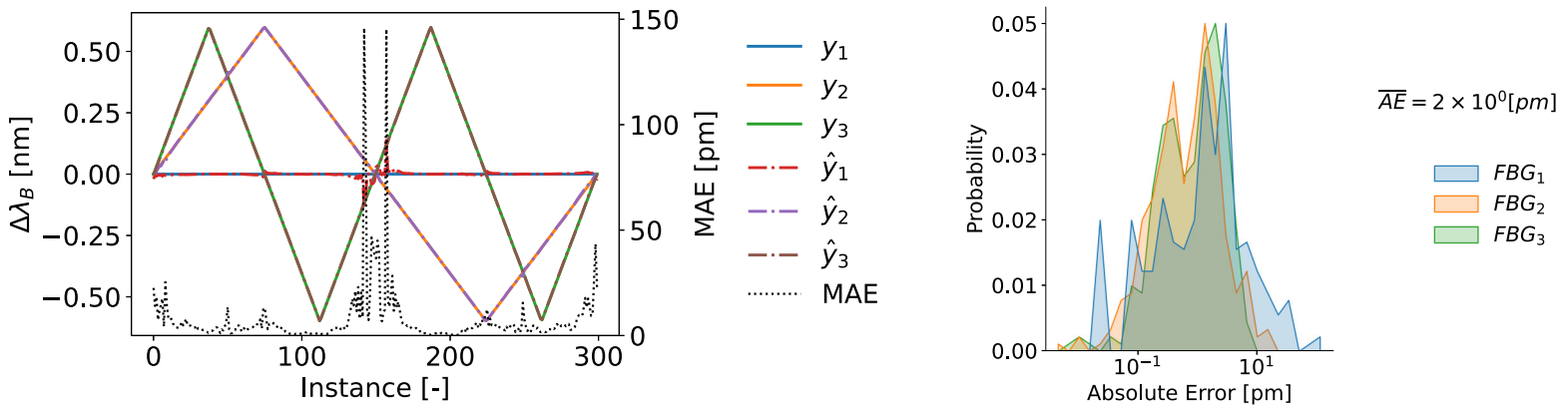


Figure 5.45: Proposed Model Sweep (left). Proposed Model Sweep Error distribution (right).

The performance of the model is displayed in figure 5.46 for the scenario of the original model tested over data drawn from the same simulation as the train data and the scenarios when there are simulation inaccuracies before and after finetuning. It can be observed that the error is significantly reduced with finetuning, achieving performance roughly on par with the original model over unmodified data.

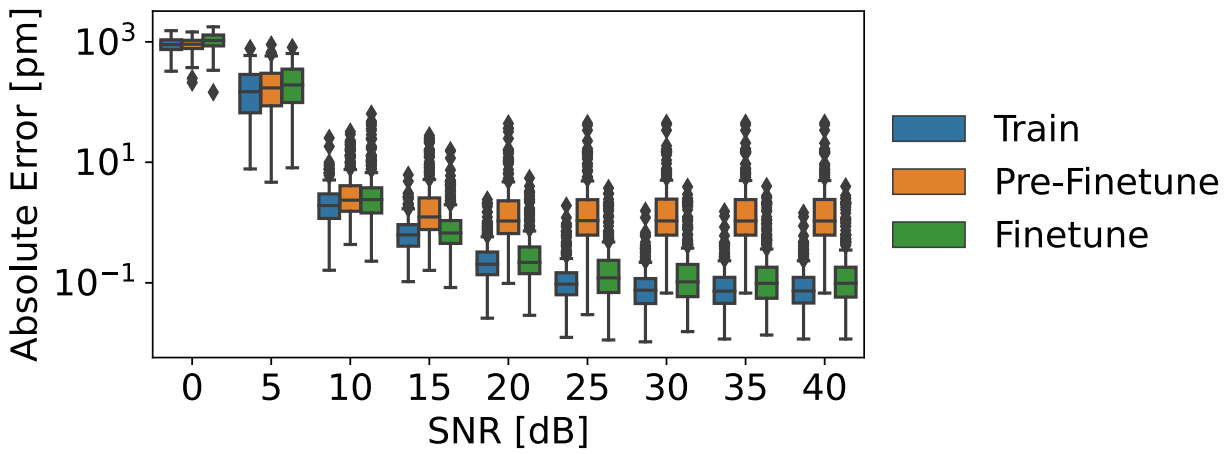


Figure 5.46: Comparison of the proposed model when train and test data match, and when they mismatch before and after finetuning.

The finetuned model is compared with the best EA and regression models in figure 5.47. It can be seen that the finetuned model outperforms the best baselines MAE by more than an order of magnitude, except for the high noise case, where SDE has a MAE of less than one order of magnitude lower. It should be noted that the model has maximum outliers of a couple of picometers.

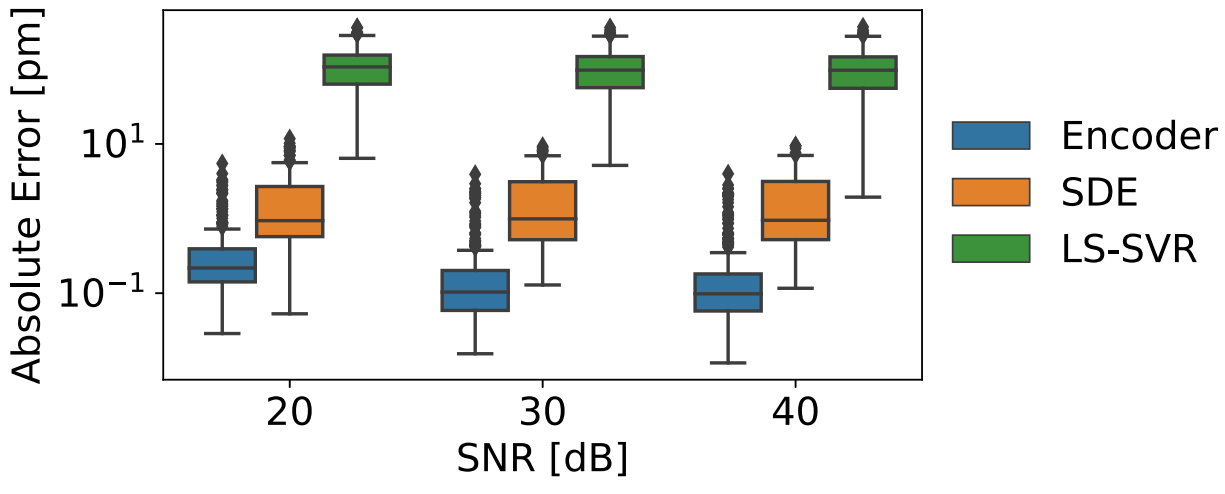


Figure 5.47: Comparison of finetuned proposed model to best baseline models.

The results displayed in figure 5.48 show that on the original domain, the pretrained model outperforms LS-SVR while having a performance on par with SDE, with marginally higher errors. In the different target domains, the baselines have significant degradation in the inference. On the other hand, the finetuned model recovers the error distribution obtained in the original domain.

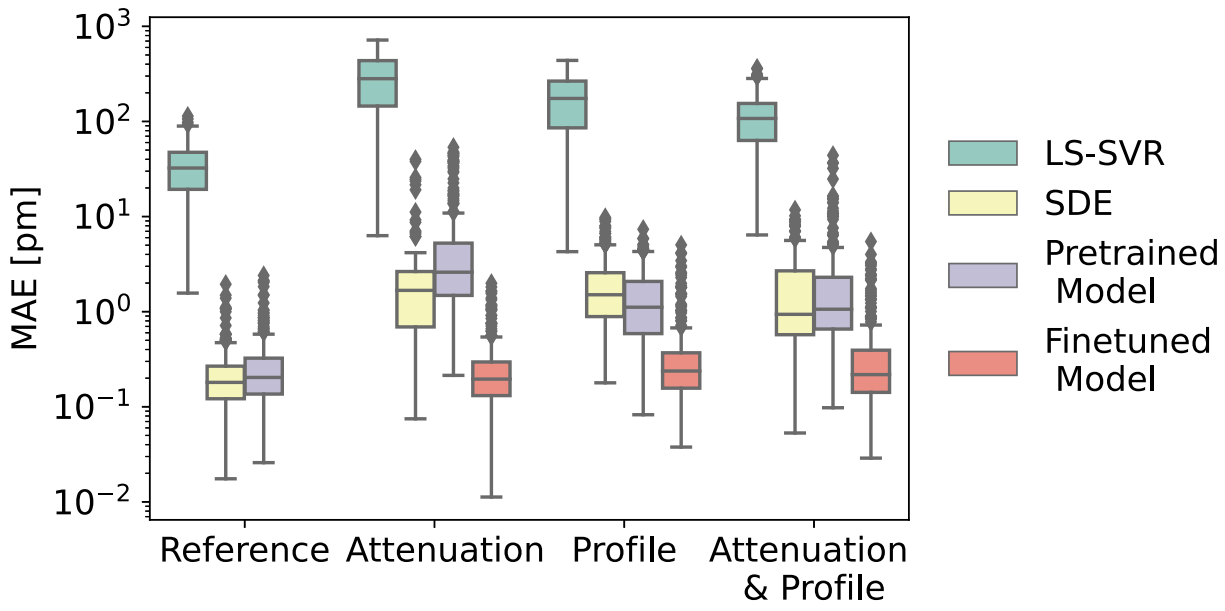


Figure 5.48: Proposal model comparison to baseline models on simulated data for the three-sensor array.

5.4 Experimental Validation

In this section, the performance of the proposed model is tested on experimental data and compared to the performance of the baseline models. The experimental setup with which the data is obtained is described in detail, as well as the design decisions behind the possible setup alternatives.

5.4.1 Experimental Setup

To construct a dataset for supervised training, the experimental setup must satisfy the following requirements. (i) Control of the spectral positions of the FBG sensors. (ii) Accurate measurement of the FBG sensors spectral positions. (iii) Measurement of the joint spectrum with an incoherent source.

For the first requirement, either the temperature or the strain of the sensors could be varied. Temperature is chosen for this purpose, as precise strain variation requires specialized equipment. Precise temperature control is achieved, with a range between temperatures of 5°C and 65°C by use of a temperature controller based on a Peltier. The design of the controller is described in detail in appendix C.

For the second requirement, it is proposed to use pairs of ”twin” FBGs inscribed with the same mask and with close peak reflectivity values. With similar characteristics, the sensors have similar sensitivities to temperature, therefore, their spectral positions for different temperatures are closely related to one another. In consequence, the spectral positions of the ”twin” pairs can be related by a linear equation by performing a linear regression of the spectral positions for different values of temperature. Then, a precise reference can be obtained for each sensor in the serial array by having a ”twin” in the same spatial position.

The optical setup of the experiment is depicted in figure 5.49. For the reference sensors a commercial FBG interrogator is used, the Luna Hyperion si255. Among the interesting characteristics of this device, is that it has 4 channels, a 10Hz sampling rate, and a wavelength range between 1500[nm] and 1600[nm] with a $10[pm]$ precision. Furthermore, it has built-in peak tracking capability.

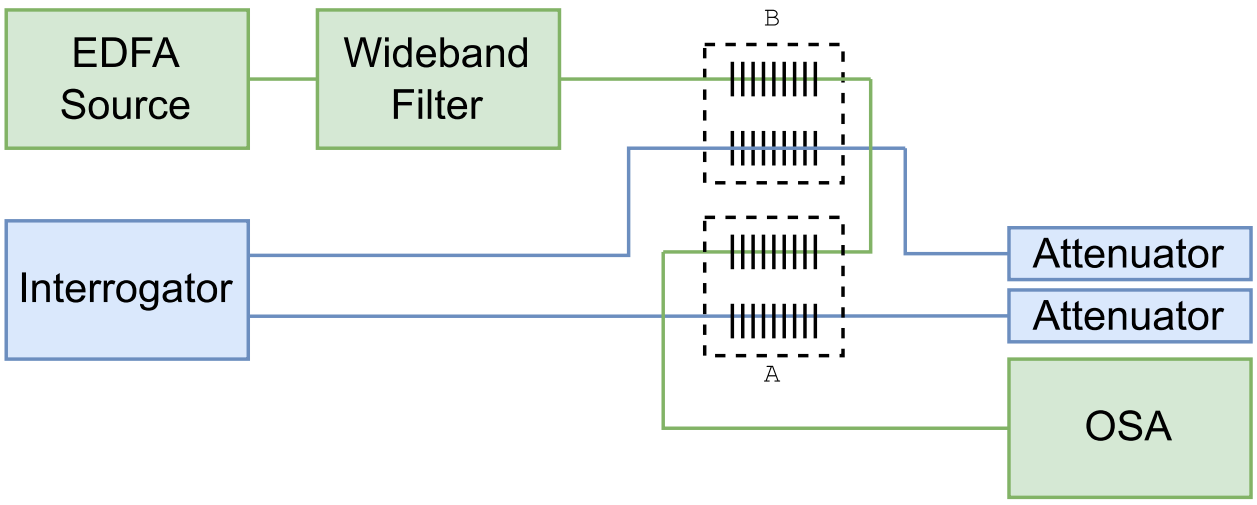


Figure 5.49: Optical components setup. The serial array and its associated equipment are depicted in green, while the reference sensors and their associated equipment are depicted in blue.

The interrogator uses as source a sweeping tunable laser, which is unsuitable for the interrogation of FBG arrays with overlapping spectra since the high-coherence source produces interference patterns. Because of this, a broadband source and an OSA are used instead to obtain the reflection spectra of the serial array.

As source, the amplified spontaneous emission (ASE) noise of an erbium-doped fiber amplifier (EDFA) is used. Using an EDFA as a broadband source is not ideal as this is not its intended use, with its main drawbacks being the fluctuation of optical power and the non-flatness of the power spectrum.

To alleviate this, instead of measuring the reflected power spectrum, the transmitted power spectrum is measured with the aim of extracting a spectral power reference from the measurements. This reference is obtained by masking the FBG peaks and using the rest of the data to obtain a background power spectrum estimate and a per-instance bias. The details of this procedure are explained in appendix A.2.

The transmission power spectrum is divided by the reference yielding the transmission spectrum T and from this, the reflection spectrum R is obtained as $R = 1 - T$. To limit the optical power going to the OSA without hindering the SNR, a wideband filter is used to restrict the light just to the necessary band.

The characteristics of the two-sensor serial array are a FWHM and a reflectivity of 0.134 nm and 58%, for the first FBG, and 0.204 nm and 93% for the second one. The details of the other optical setup components are described in table 5.6.

Table 5.6

Component	Model	Details
EDFA	Thorlabs Fiber Amplifier	
Interrogator	Luna Hyperion si255	
Wideband Filter	DK Photonics Filter	$BW = 2[nm]$, $\lambda_0 = 1550[nm]$
Attenuator	Generic	$Attenuation = 10[dB]$
OSA	Apex AP2683A	$Resolution = 1.12[pm]$

The experiments consist of one FBG pair set to a constant temperature while the other FBG pair is smoothly varied from on of the temperature range limit to the opposite. This yields all the combinations of the spectral distance between the FBGs in the serial array. The experiment details are described in table 5.7.

Table 5.7: Description of experiments.

Experiment number	Constant sensor	Temperature [°C]			Change Rate [°C/sec]
		Const.	Init.	End	
1	A	32.5	5	65	1/300
2	A	32.5	65	5	1/300
3	B	40	5	65	1/300
4	B	40	65	5	1/300

5.4.2 Experimental Results

The source domain corresponds to the pseudo-simulation, based on shifted measurements of the individual spectra from the FBG sensors of the array, as described in 2.9. This method of simulation is used as it minimizes the simulation inaccuracy, i.e. it minimizes the differences between source and target domain. The hyperparameters for the CNN-AE obtained through simulation are maintained, except for the parametrization of the transposed convolution kernel, where the *centered* parametrization is used to address non-symmetrical spectral profiles (see section 5.1.4).

The experimental data is partitioned into train and test subsets, with a test proportion of 10%. The train dataset is used for finetuning the pretrained model, while the test dataset is used to evaluate and compare with the baselines.

Before finetuning, the performance of the pre-trained model is evaluated on the test data, the results for this scenario are shown in figure 5.50. As can be seen, the model performs poorly, with prominent peaks of values as high as $250[pm]$.

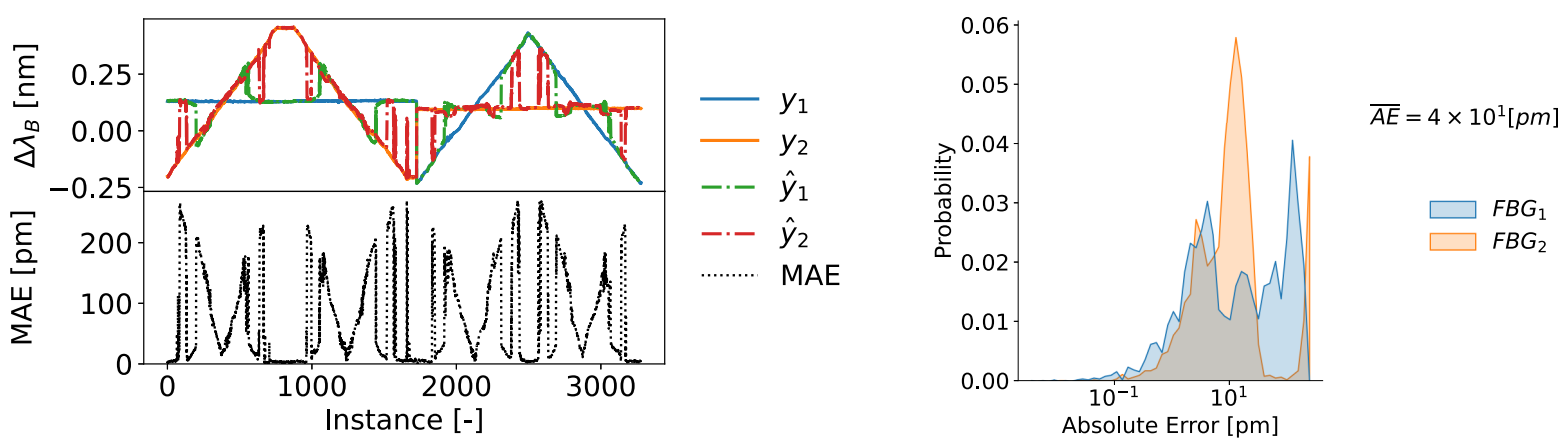


Figure 5.50: Pretrained model inference on test data (left). Pretrained model error distribution on test data (right).

The model is finetuned as described in section 5.2.2. The inference performance over the test data is shown in figure 5.51. We can note that the performance improved substantially but there are still significantly prominent peaks around the crossing point.

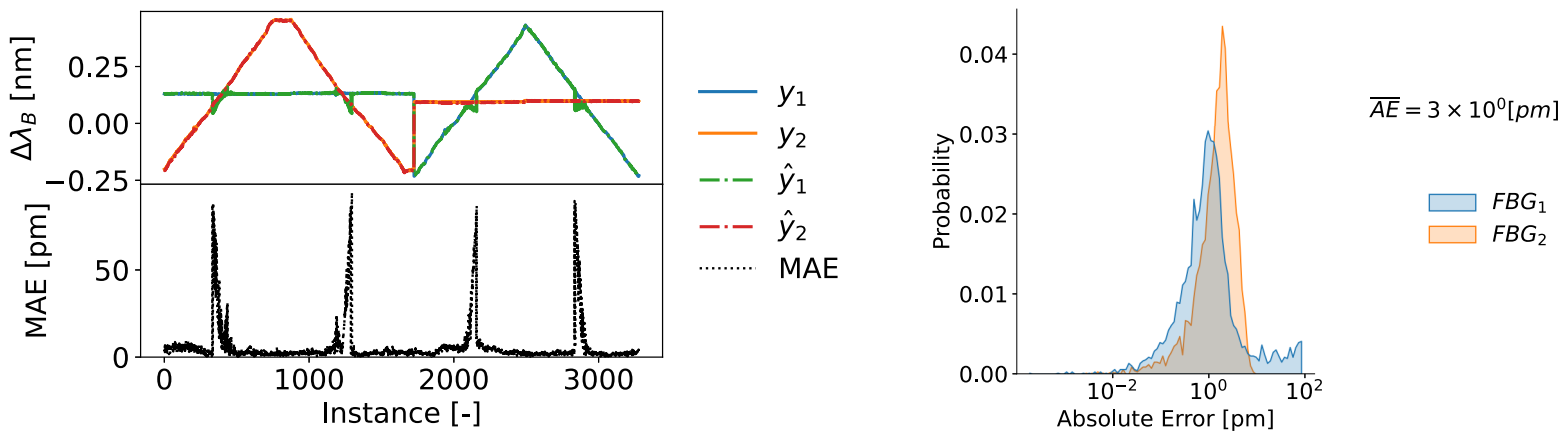


Figure 5.51: Finetuned model inference on test data (left). Finetuned model error distribution on test data (right).

The box plots of the MAE are shown for the simulation (source domain) and experimental (target domain) scenarios in figure 5.52. In the simulation scenario the pretrained model outperforms LS-SVR while displaying a performance close to SDE, and in the experimental scenario the finetuned model closely matches the performance of SDE.

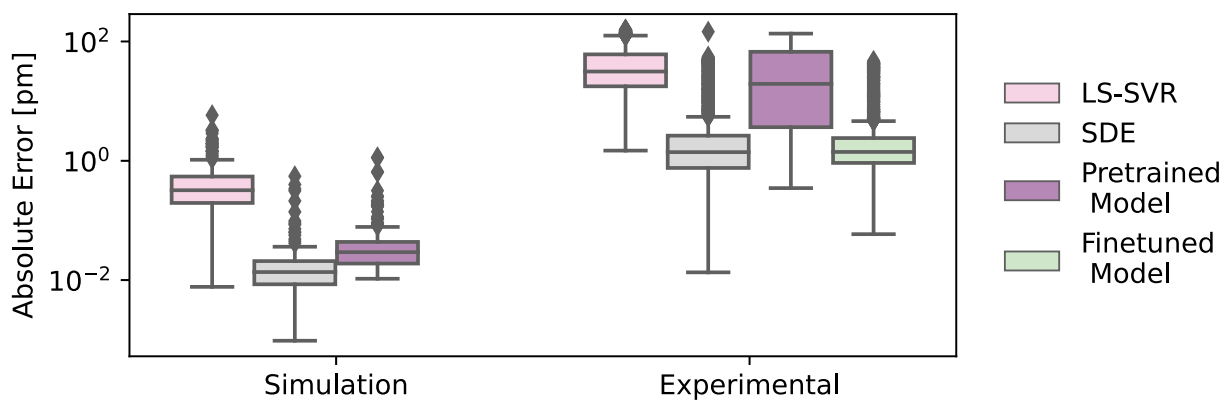


Figure 5.52: Proposed model comparison to baseline models on simulated data and experimental data.

The obtained results suggest that the particular setup of the array is not suitable to obtain adequate inference around the crossing point. The spectral shapes have low sidelobes as can be seen from figure 5.53 and the measurements of the serial array have a high level of noise which further conceal the sidelobes. If instead, the sidelobes were not covered in this manner, they could be useful to determine the spectral positions of the sensors.

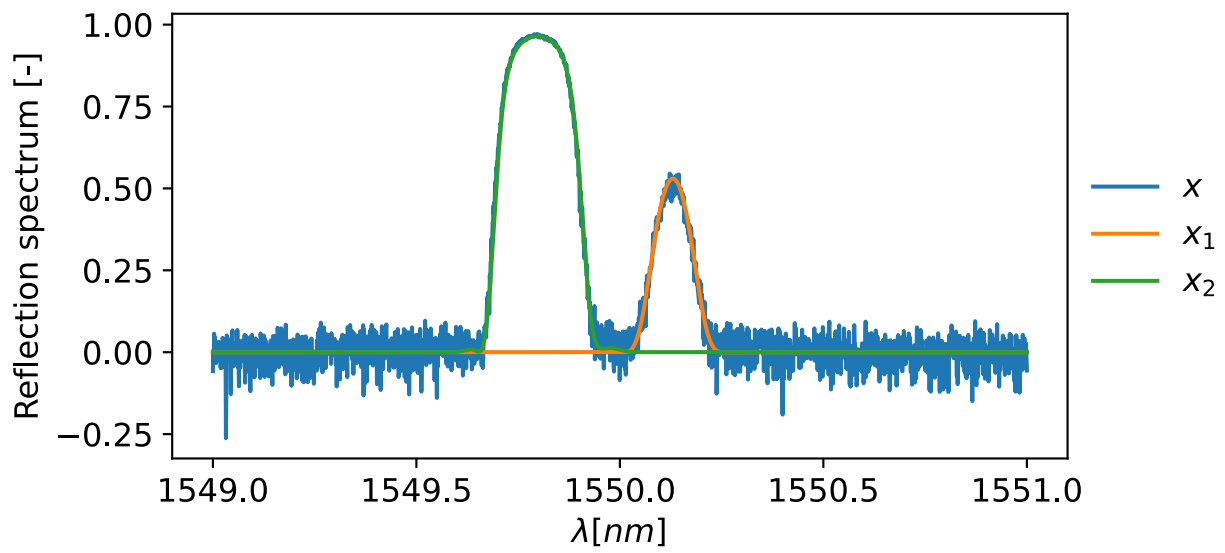


Figure 5.53: Single measurement of the serial array with superposed individual measured spectra measurements with spectral shifts.

In order to evaluate the impact of noise with these particular spectral shapes, the model was pretrained with simulated data of Gaussian simulation profiles with the characteristics of the sensors and then finetuned with the pseudo-simulated dataset. It is found that the model learns an adequate inference when there is no noise, as shown in figure 5.54.

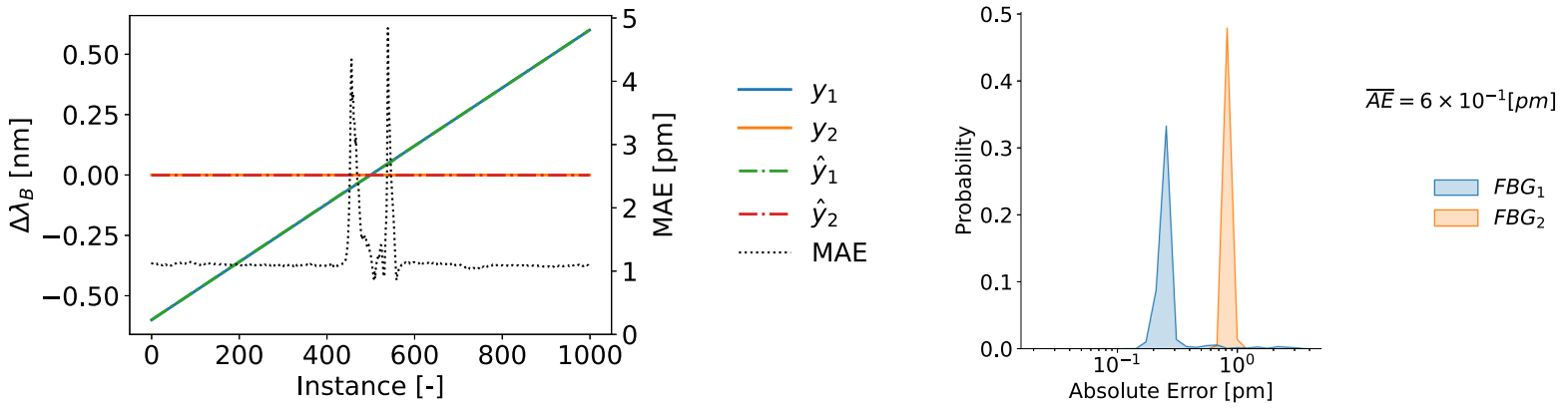


Figure 5.54: Finetuned model inference on test data (left). Finetuned model error distribution on test data (right).

Then, noise is added to the data and it is found that only very low levels of noise allowed the model to learn an adequate inference around the crossing point. Concretely, the SNR level should be greater than $30dB$. The result for finetuning with this level of noise is displayed in figure 5.55

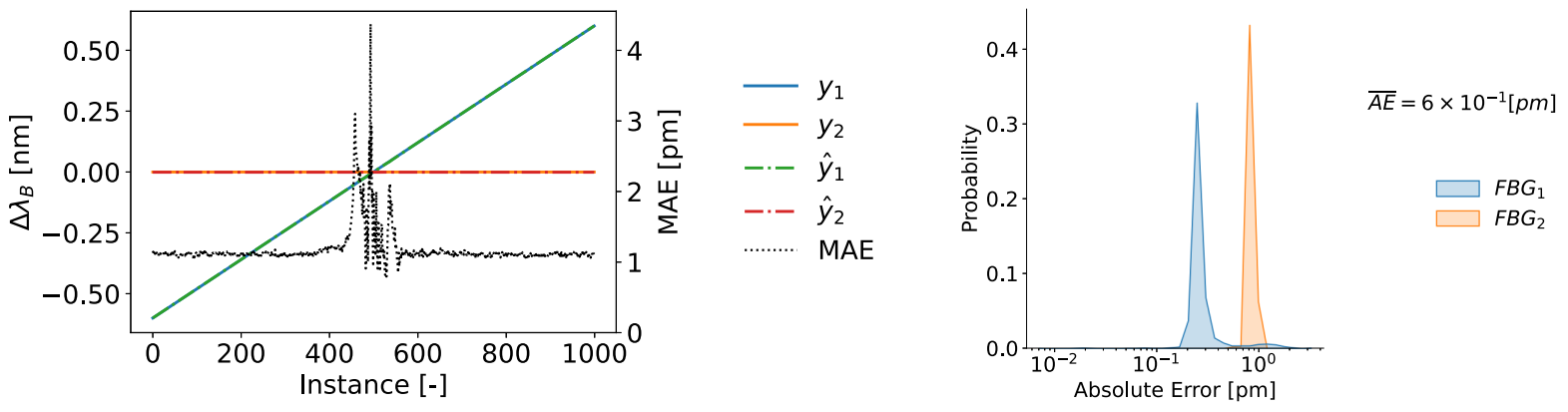


Figure 5.55: Finetuned model inference on test data with noise(left). Finetuned model error distribution on test data with noise(right).

From figure 5.56 it can be observed that after finetuning the performance of the proposed model is improved significantly, with a small number of outliers at a level of a few picometers. It can also be seen that when finetuning with data with an SNR of $30dB$ the error distribution is almost identical to the noiseless case. It was found that for lower SNR levels the performance of the finetuned model degrades to levels seen for the experimental dataset.

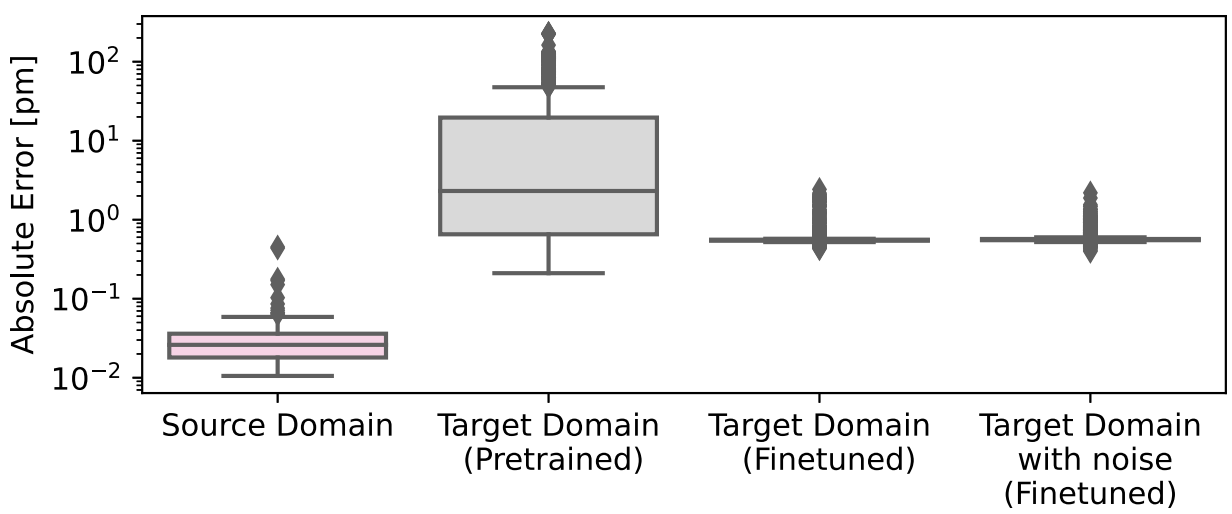


Figure 5.56: Performance comparison of the pretrained and the finetuned model on pseudo-simulated data. The target domain has no noise, except when it is specified.

Based on the results, it can be concluded that the low performance of the methods can be explained by the low SNR of the experimental setup and the specific spectral shapes of the FBGs, which make it challenging to perform inference when there is high overlap.

5.5 Discussion

In this work, a CNN AE is proposed to solve the problem of SWDM with the capability of training on simulated data and adaptation to real data by unsupervised domain adaptation. The model performance and its capability for unsupervised domain adaptation is evaluated in different scenarios and compared to baseline methods previously proposed in the literature.

Finetuning proved to give adequate results on simulated data even when significant simulation inaccuracies are introduced. The case of a two-sensor array was evaluated, giving promising results with consistent subpicometer accuracy. For the two-sensor and three-sensor cases, finetuning proved effective, reducing the MAE significantly and retrieving a performance on par with the one obtained on the original domain through supervised means. It should be noted that the chosen hyperparameters were optimized for the two-sensor case. In consequence, these results are not conclusive and should be regarded as early tests for the three-sensor case. For more conclusive results, a set of optimal hyperparameters should be found for this scenario.

Experiments on real data demonstrate that the finetuned model matches the best baseline that uses a simulation based on accurate individual spectra measurements to evaluate its candidates. However, the performance was significantly degraded with respect to the simulation, which corresponds to the original domain. Experiments on simulated data suggest that the particular spectral shapes with the SNR of the measurements are not suitable for adequate SWDM demultiplexation with the proposed method or the baseline methods.

Chapter 6

Conclusion

Among the main contributions of this work, it presents a framework for the simulation of FBG arrays. In addition to the typical gaussian approximation of reflection spectra, a simulation for the uniform modulation case is included, which is based on a parametrization of the physical properties by spectral characteristics such as peak reflectivity and bandwidth. A simulation procedure for the parallel topology is also provided, as well as a simulation for the more challenging serial topology. The simulation for the serial topology is particularly noteworthy because it is based on the Transfer-Matrix Formalism and accurately accounts for incoherent gaps between sensors, which to the best of our knowledge has not been done before.

A comprehensive literature review of SWDM is presented, which includes a large number of proposed algorithms. Implementations of multiple algorithms from the literature, both ML and EA approaches were constructed, to be used as baselines. Evaluating these baselines with the simulation framework allows for a fair comparison between them. From the literature, a reliance on simulated data is observed. For the ML approach simulation are required to confection a dataset necessary for training. On the other hand, for the EA approach, imulations are required for the evaluation of candidates. This brings forth the problem of simulation inaccuracies, where differences between simulation and real data translate to decreased performance.

This work addresses the simulation inaccuracies by performing unsupervised domain adaptation on an ML model. In this setting, the model is first trained on a dataset and then retrained in an unsupervised manner on a different target dataset, to adapt the model to the target distribution of data. In this context, we term the training on simulated data *pretraining* and the subsequent unsupervised training on real data *finetuning*. This work proposes the use of an CNN-AE with the necessary inductive biases to align the encoding of the latent variable between the regression objective and the reconstruction objective, so that the pretraining and posterior finetuning lead to better inference on the real data. The main characteristics of the proposal are a Dilated CNN encoder with no pooling, an ad-hoc regularization of the latent variable, and a simple decoder that follows closely the simulation procedure.

The performance of the model and its ability to adapt to new domains without supervision were tested in various situations and compared to previously published baseline methods. Finetuning was found to be effective on simulated data, even when the simulation inaccuracy is simulated by perturbations in simulation parameters. When using a two-sensor array, finetuning resulted in consistently high accuracy at the subpicometer level. While using a three-sensor array also showed improvement with finetuning, it did not consistently reach subpicometer accuracy, especially when the sensors crossed over. It is worth noting that the hyperparameters were optimized for the two-sensor case, so these results should be considered preliminary for the three-sensor case. To get more reliable results, a set of optimal hyperparameters should be identified for this three-sensor scenario. The experiments on real data indicated that the success of the SWDM method depends on more than just being able to differentiate between peaks of reflectivity. In particular, the SNR of the experimental data and the spectral shape of each sensor was found to be critical for achieving accurate results.

The necessary code for the replication of the results of this work is available at `https://github.com/grudloff/fbg_S-WDM`. The repository contains the code for the simulation, the baseline algorithms as well as the proposal. This work intends to provide a base to be used as a common benchmark and allow future research to focus on improving the methods. Further studies are still required to analyze possible approaches to improve the finetuned AE model by overcoming some limitations presumably imposed by the FBG spectral shapes and measurement SNR.

Bibliography

[1] A. Othonos, K. Kalli, D. Pureur, and A. Mugnier, "Fibre bragg gratings," in *Wavelength Filters in Fibre Optics*, Springer, 2006, pp. 189–269.

[2] R. Kashyap, *Fiber bragg gratings*. Academic press, 2009.

[3] J. T. Bagnara, P. J. Fernandez, and R. Fujii, "On the blue coloration of vertebrates," *Pigment Cell Research*, vol. 20, no. 1, pp. 14–26, 2007.

[4] G. S. Smith, "Structural color of morpho butterflies," *American Journal of Physics*, vol. 77, no. 11, pp. 1010–1019, 2009.

[5] C. A. Tippets, Y. Fu, A.-M. Jackson, E. U. Donev, and R. Lopez, "Reproduction and optical analysis of morpho-inspired polymeric nanostructures," *Journal of Optics*, vol. 18, no. 6, p. 065 105, 2016.

[6] J. Teyssier, S. V. Saenko, D. Van Der Marel, and M. C. Milinkovitch, "Photonic crystals cause active colour change in chameleons," *Nature communications*, vol. 6, no. 1, pp. 1–7, 2015.

[7] B. Osting, "Bragg structure and the first spectral gap," *Applied Mathematics Letters*, vol. 25, no. 11, pp. 1926–1930, 2012.

[8] M. Born and E. Wolf, *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*. Elsevier, 2013.

[9] T. Erdogan, "Fiber grating spectra," *Journal of lightwave technology*, vol. 15, no. 8, pp. 1277–1294, 1997.

[10] K. O. Hill and G. Meltz, "Fiber bragg grating technology fundamentals and overview," *Journal of lightwave technology*, vol. 15, no. 8, pp. 1263–1276, 1997.

[11] A. Yariv, "Coupled-mode theory for guided-wave optics," *IEEE Journal of Quantum Electronics*, vol. 9, no. 9, pp. 919–933, 1973.

[12] H. Kogelnik, "Theory of optical waveguides," in *Guided-wave optoelectronics*, Springer, 1988, pp. 7–88.

[13] T. Erdogan, "Cladding-mode resonances in short-and long-period fiber grating filters," *JOSA A*, vol. 14, no. 8, pp. 1760–1773, 1997.

[14] J. He, B. Xu, X. Xu, C. Liao, and Y. Wang, "Review of femtosecond-laser-inscribed fiber bragg gratings: Fabrication technologies and sensing applications," *Photonic Sensors*, vol. 11, no. 2, pp. 203–226, 2021.

[15] D. Tosi, "Review and analysis of peak tracking techniques for fiber bragg grating sensors," *Sensors*, vol. 17, no. 10, p. 2368, 2017.

[16] Y.-J. Rao, "In-fibre bragg grating sensors," *Measurement science and technology*, vol. 8, no. 4, p. 355, 1997.

[17] B. Lee and Y. Jeong, "Interrogation techniques for fiber grating sensors and the theory of fiber gratings," in *Fiber Optic Sensors*, CRC Press, 2017, pp. 253–332.

[18] K. Grattan and T. Sun, "Fiber optic sensor technology: An overview," *Sensors and Actuators A: Physical*, vol. 82, no. 1-3, pp. 40–61, 2000.

[19] M. Fajkus, I. Navruz, S. Kepak, *et al.*, "Capacity of wavelength and time division multiplexing for quasi-distributed measurement using fiber bragg gratings," *Advances in Electrical and Electronic Engineering*, vol. 13, no. 5, pp. 575–582, 2015.

[20] D. Hwang, D.-C. Seo, I.-B. Kwon, and Y. Chung, "Restoration of reflection spectra in a serial fbg sensor array of a wdm/tdm measurement system," *Sensors*, vol. 12, no. 9, pp. 12 836–12 843, 2012.

[21] K. Koo, A. Tveten, and S. Vohra, "Dense wavelength division multiplexing of fibre bragg grating sensors using cdma," *Electronics Letters*, vol. 35, no. 2, pp. 165–167, 1999.

[22] J. Gong, J. MacAlpine, C. Chan, W. Jin, M. Zhang, and Y. Liao, "A novel wavelength detection technique for fiber bragg grating sensors," *IEEE Photonics Technology Letters*, vol. 14, no. 5, pp. 678–680, 2002.

[23] Q.-X. Zhou, H. Jiang, Y.-T. Lin, and J. Chen, "Application of distributed estimation algorithm in wavelength demodulation of overlapping spectra of fiber bragg grating sensor networks," in *2019 Chinese Control Conference (CCC)*, IEEE, 2019, pp. 6320–6324.

[24] H. Jiang, J. Chen, T. Liu, and W. Huang, "A novel wavelength detection technique of overlapping spectra in the serial wdm fbg sensor network," *Sensors and Actuators A: Physical*, vol. 198, pp. 31–34, 2013.

[25] C. C. Katsidis and D. I. Siapkas, "General transfer-matrix method for optical multilayer systems with coherent, partially coherent, and incoherent interference," *Applied optics*, vol. 41, no. 19, pp. 3978–3987, 2002.

[26] M. C. Troparevsky, A. S. Sabau, A. R. Lupini, and Z. Zhang, "Transfer-matrix formalism for the calculation of optical response in multilayer systems: From coherent to incoherent interference," *Optics express*, vol. 18, no. 24, pp. 24 715–24 721, 2010.

[27] B. Harbecke, "Coherent and incoherent reflection and transmission of multilayer structures," *Applied Physics B*, vol. 39, no. 3, pp. 165–170, 1986.

[28] R. Santbergen, A. H. Smets, and M. Zeman, "Optical model for multilayer structures with coherent, partly coherent and incoherent layers," *Optics express*, vol. 21, no. 102, A262–A267, 2013.

[29] L. Zhang, Y. Liu, J. Williams, and I. Bennion, "Enhanced fbg strain sensing multiplexing capacity using combination of intensity and wavelength dual-coding technique," *IEEE Photonics Technology Letters*, vol. 11, no. 12, pp. 1638–1640, 1999.

[30] C. A. Triana, D. Pastor, and M. Varón, "Enhancing the multiplexing capabilities of sensing networks using spectrally encoded fiber bragg grating sensors," *Journal of Lightwave Technology*, vol. 34, pp. 4466–4472, 2016.

[31] A. Triana, D. Pastor, and M. Varón, "A code division design strategy for multiplexing fiber bragg grating sensing networks," *Sensors (Basel, Switzerland)*, vol. 17, 2017.

[32] P. Childs, A. C. Wong, B. Yan, M. Li, and G.-D. Peng, "A review of spectrally coded multiplexing techniques for fibre grating sensor systems," *Measurement Science and Technology*, vol. 21, no. 9, p. 094 007, 2010.

[33] O. Belai, L. Frumin, E. Podivilov, and D. Shapiro, "Efficient numerical method of the fiber bragg grating synthesis," *JOSA B*, vol. 24, no. 7, pp. 1451–1457, 2007.

[34] C. Chan, C. Shi, J. Gong, W. Jin, and M. Demokan, "Enhancement of the measurement range of fbg sensors in a wdm network using a minimum variance shift technique coupled with amplitude-wavelength dual coding," *Optics communications*, vol. 215, no. 4-6, pp. 289–294, 2003.

[35] G. Guo, D. Hackney, M. Pankow, and K. Peters, "Interrogation of a spectral profile division multiplexed fbg sensor network using a modified particle swarm optimization method," *Measurement Science and Technology*, vol. 28, no. 5, p. 055 204, 2017.

[36] Y. C. Manie, J.-W. Li, P.-C. Peng, R.-K. Shiu, Y.-Y. Chen, and Y.-T. Hsu, "Using a machine learning algorithm integrated with data de-noising techniques to optimize the multipoint sensor network," *Sensors*, vol. 20, no. 4, p. 1070, 2020.

[37] S. Luke, *Essentials of metaheuristics*. Lulu Raleigh, 2013, vol. 2.

[38] C. Shi, C. Chan, W. Jin, Y. Liao, Y. Zhou, and M. Demokan, "Improving the performance of a fbg sensor network using a genetic algorithm," *Sensors and Actuators A: Physical*, vol. 107, no. 1, pp. 57–61, 2003.

[39] C. Shi, N. Zeng, C. Chan, Y. Liao, W. Jin, and L. Zhang, "Improving the performance of fbg sensors in a wdm network using a simulated annealing technique," *IEEE Photonics Technology Letters*, vol. 16, no. 1, pp. 227–229, 2004.

[40] J. Liang, C. Chan, V. Huang, and P. Suganthan, "Improving the performance of a fbg sensor network using a novel dynamic multi-swarm particle swarm optimizer," in *Sensors for Harsh Environments II*, SPIE, vol. 5998, 2005, pp. 191–197.

[41] J. Liang, P. Suganthan, C. Chan, and V. Huang, "Wavelength detection in fbg sensor network using tree search dms-pso," *IEEE Photonics Technology Letters*, vol. 18, no. 12, pp. 1305–1307, 2006.

[42] D. Liu, K. Tang, Z. Yang, and D. Liu, "A fiber bragg grating sensor network using an improved differential evolution algorithm," *IEEE Photonics Technology Letters*, vol. 23, no. 19, pp. 1385–1387, 2011.

[43] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1110–1116. DOI: 10.1109/CEC.2008.4630935.

[44] Y. Qi, C. Li, P. Jiang, C. Jia, Y. Liu, and Q. Zhang, "Research on demodulation of fbgs sensor network based on pso-sa algorithm," *Optik*, vol. 164, pp. 647–653, 2018.

[45] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[46] N. Zeng, C. Shi, C. Chan, *et al.*, "Enhancement of the measurement range of fbg sensors in a wdm network: A self-organizing network solution," *Sensors and Actuators A: Physical*, vol. 118, no. 2, pp. 233–237, 2005.

[47] H. Jiang, J. Chen, and T. Liu, "Wavelength detection in spectrally overlapped fbg sensor network using extreme learning machine," *IEEE Photonics Technology Letters*, vol. 26, no. 20, pp. 2031–2034, 2014.

[48] Y. Manie, R.-K. Shiu, P.-C. Peng, *et al.*, "Intensity and wavelength division multiplexing fbg sensor system using a raman amplifier and extreme learning machine," *Journal of Sensors*, vol. 2018, pp. 1–11, Sep. 2018. DOI: 10.1155/2018/7323149.

[49] J. Chen, H. Jiang, T. Liu, and X. Fu, "Wavelength detection in fbg sensor networks using least squares support vector regression," *Journal of Optics*, vol. 16, no. 4, p. 045 402, 2014.

[50] Y. Wang, J. Chen, and H. Jiang, "Wavelength demodulation of overlapping spectra in fbg sensor network based on deep neural network," in *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, IEEE, 2020, pp. 919–924.

[51] H. Jiang, Q. Zeng, J. Chen, *et al.*, "Wavelength detection of model-sharing fiber bragg grating sensor networks using long short-term memory neural network," *Optics express*, vol. 27, no. 15, pp. 20 583–20 596, 2019.

[52] Y. C. Manie, J.-W. Li, P.-C. Peng, R.-K. Shiu, Y.-Y. Chen, and Y.-T. Hsu, "Using a machine learning algorithm integrated with data de-noising techniques to optimize the multipoint sensor network," *Sensors*, vol. 20, no. 4, p. 1070, 2020.

[53] Y. C. Manie, P.-C. Peng, R.-K. Shiu, *et al.*, "Enhancement of the multiplexing capacity and measurement accuracy of fbg sensor system using iwdm technique and deep learning algorithm," *Journal of Lightwave Technology*, vol. 38, no. 6, pp. 1589–1603, 2020.

[54] B. Li, Z.-W. Tan, P. P. Shum, *et al.*, "Robust convolutional neural network model for wavelength detection in overlapping fiber bragg grating sensor network," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, IEEE, 2020, pp. 1–3.

[55] B. Li, Z.-W. Tan, P. P. Shum, C. Wang, Y. Zheng, and L. jie Wong, "Dilated convolutional neural networks for fiber bragg grating signal demodulation," *Optics Express*, vol. 29, no. 5, pp. 7110–7123, 2021.

[56] P.-H. Chiu, Y.-S. Lin, Y. C. Manie, J.-W. Li, J.-H. Lin, and P.-C. Peng, "Intensity and wavelength-division multiplexing fiber sensor interrogation using a combination of autoencoder pre-trained convolution neural network and differential evolution algorithm," *IEEE Photonics Journal*, vol. 13, no. 1, pp. 1–9, 2021.

[57] F. López-Muñoz, J. Boya, and C. Alamo, "Neuron theory, the cornerstone of neuroscience, on the centenary of the nobel prize award to santiago ramón y cajal," *Brain research bulletin*, vol. 70, no. 4-6, pp. 391–405, 2006.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[59] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[60] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 4. Springer, 2006, vol. 4.

[61] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen, "The modern mathematics of deep learning," *CoRR*, vol. abs/2105.04026, 2021. arXiv: 2105.04026.

[62] C. Olah, *Neural networks, manifolds, and topology*, Apr. 2014. [Online]. Available: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/.

[63] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, Atlanta, Georgia, USA, vol. 30, 2013, p. 3.

[64] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[65] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *ArXiv*, vol. abs/1710.05941, 2018.

[66] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *CoRR*, vol. abs/1908.08681, 2019. arXiv: 1908.08681. [Online]. Available: http://arxiv.org/abs/1908.08681.

[67] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181.

[68] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[69] C. Olah, *Understanding lstm networks*, Aug. 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[70] S. Mallat, "Group invariant scattering," *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, 2012.

[71] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[72] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[73] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[75] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[76] A. v. d. Oord, S. Dieleman, H. Zen, *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[77] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[78] S. Mallat, *A wavelet tour of signal processing*. Elsevier, 1999.

[79] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[80] G. Goh, "Why momentum really works," *Distill*, 2017. DOI: 10.23915/distill.00006. [Online]. Available: http://distill.pub/2017/momentum.

[81] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[82] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7.

[83] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[84] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[85] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[86] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[87] L. N. Smith and N. Topin, "Super-convergence: Very fast training of residual networks using large learning rates. corr abs/1708.07120 (2017)," *arXiv preprint arXiv:1708.07120*, 2017.

[88] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.

[89] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1– learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[90] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[91] V. Niculae, *Optimizing with constraints: Reparametrization and geometry.* Sep. 2020. [Online]. Available: `https://vene.ro/blog/mirror-descent.html#generalizing-the-projected-gradient-method-with-divergences`.

[92] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018. [Online]. Available: `https://openreview.net/forum?id=r1Ddp1-Rb`.

[93] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[94] G. Wilson and D. J. Cook, "A survey of unsupervised deep domain adaptation," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 5, Jul. 2020, ISSN: 2157-6904. DOI: `10.1145/3400066`. [Online]. Available: `https://doi.org/10.1145/3400066`.

[95] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[96] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[97] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

[98] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[99] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[100] N. Jiang, W. Rong, B. Peng, Y. Nie, and Z. Xiong, "An empirical analysis of different sparse penalties for autoencoder in unsupervised feature learning," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280568.

[101] G. Rodrıguez, "Smoothing and non-parametric regression," *Princeton University*, 2001.

[102] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," *arXiv preprint arXiv:1905.11881*, 2019.

[103] M. A. Coletti, E. O. Scott, and J. K. Bassett, "Library for evolutionary algorithms in python (leap)," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20, Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1571–1579, ISBN: 9781450371278. DOI: 10.1145/3377929.3398147. [Online]. Available: https://doi.org/10.1145/3377929.3398147.

[104] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Appendix A

Miscellaneous

A.1 Saturation Function

As described in section 2.2 the FWHM bandwidth of an FBG is given by

$$\Delta\lambda = S(\zeta)\Delta\lambda_0$$

where $\Delta\lambda_0$ is the mainlobe bandwidth as measured between its nulls and $\lambda = S(\zeta)$ is an unknown function dependent on ζ. We choose to approximate this function as a weighted sum of powers of ζ that is then passed through a saturation function. In particular we use only the zeroth and first powers and use the $tanh$ function described in section 4.5, this is described by

$$S(\zeta) = sigmoid(\Omega_0 + \Omega_1\zeta)$$

where $\Omega = \{\Omega_0, \Omega_1\}$ is a weight vector. This parameter is learned through bayesian optimization by simulating batches with varying saturations and optimizing the difference between $S(\zeta)\Delta\lambda_0$ and the actual measured $\Delta\lambda$. In our tests we found this approximation to be sufficiently good, achieving relative errors in the order of 10^{-2}. Adding more powers of ζ did not improve the performance. The learned weight vector is given by[22]

$$\Omega = \{-0.6050385590422893, 0.8574601694330704\}$$

A.2 Reflectance Computation

As described in section 5.4, the transmitted spectrum is measured, instead of the reflection spectrum, to obtain an input power spectrum reference together with the reflection profiles of each spectral measurement. This allows for addressing the non-flatness and time-varying properties of the light source that was used. In this section, we describe the process by which the reflection spectra are estimated from the measured transmission spectra. This process involves several steps, which are summarized below:

1. Obtain a rough reference estimate of the input power spectrum by fitting a line to

[22] A large number of decimal digits are given for reproducibility purposes.

each measured spectrum excluding the dips corresponding to the reflection spectra of each sensor. An example of the obtained rough reference is displayed in figure A.1 on the graph of the left.

2. Calculate rough reflection spectra and an estimation of the peak position from each spectral measurement using the rough reference estimate.

3. Use the peak position data where there is no overlap to calculate a linear regression with the peak position data of the corresponding twin sensors to obtain the peak positions when there is overlap between the sensors.

4. Mask the transmission power spectrum with the peak position estimations, and calculate a reference power profile and a per spectral measurement bias from the non-masked data.

5. Obtain refined reference power spectra for each spectral measurement using the reference power profile and the per spectral measurement bias. This is illustrated as an example in figure A.1 on the graph to the right.

6. Finally, the refined reflection spectra are calculated from the transmission power spectrum and the refined power spectra reference.

The details of each step are described in more detail below.

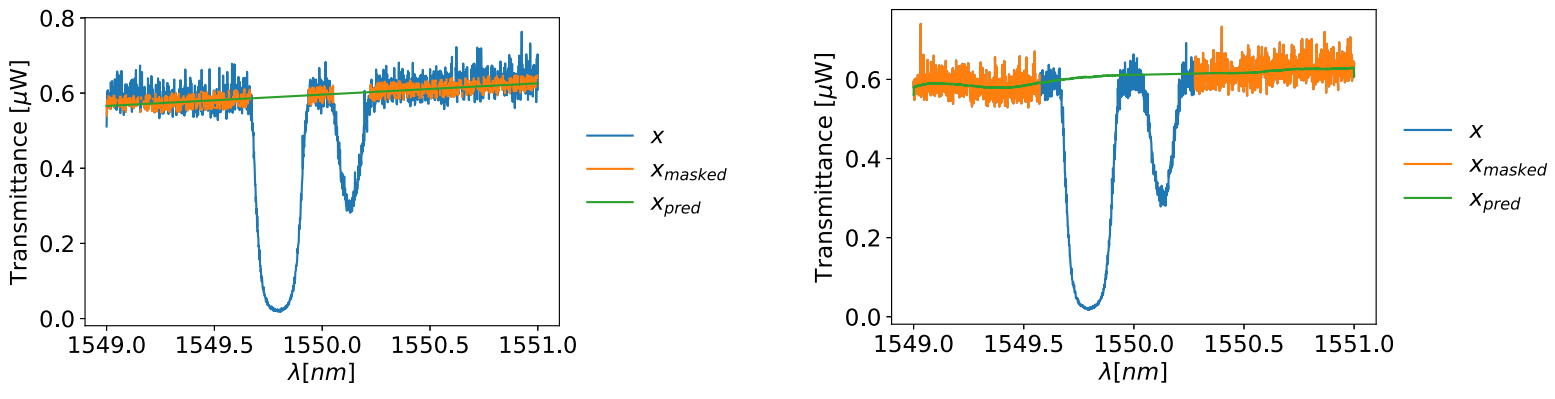


Figure A.1: Rough estimate (left). Refined estimate (right). x is an instance of spectral measurement, x_{masked} is the masked spectral measurement displaying the data that is used for the reference estimate and x_{pred} is the corresponding reference estimate.

To begin, algorithm 4 is used to find a linear estimate of the reference power. The algorithm consists in iteratively fitting a line to each spectral measurement and excluding, for the next iteration, the proportion of data further from the fitted line. The inputs of the algorithm are the matrix containing the spectral measurements X, the portion of data excluded in each iteration $\alpha \in [0, 1]$, and the number of iterations $n_{iterations}$. From the result of this algorithm, the transmittance spectrum T is obtained by dividing the transmission power spectrum by the reference. From the transmittance spectrum the reflection spectrum R is obtained as $R = T - 1$.

Algorithm 4 Linear reference iterative estimation

Require: $X, \alpha, n_{iteration}$
  $lines \leftarrow fit\_lines(X)$ ▷ fit line to each spectral measurement
  $M \leftarrow len(X)$
  for $i \in [1, \cdots, M]$ **do**
    for $j \in [1, \cdots, n_{iteration}]$ **do**
      $v \leftarrow X[i]$
      $line \leftarrow lines[i]$
      $v \leftarrow mask(v, line, \alpha)$ ▷ mask portion α from v with greatest distance to line
      $line \leftarrow fit\_line(v)$
    end for
  end for
  return $lines$

The peak estimates are obtained from the rough reflection spectrum by using the `find_peaks` function from the `scipy` library. This estimation is improved by re-estimating the peak from the center of mass of each peak.

There is a portion of data where the peaks overlap, and as a consequence, there is no clear estimate of the peak position. The estimated peaks that are sufficiently apart to have no overlap are used to obtain a regression with respect to the peaks of the twin FBG sensors, obtained from the interrogator. From this regression, an estimation of the peak positions is computed.

Then, the peak estimations from each sensor are used to obtain the refined reflection spectrum. With this objective, the peak estimation and FWHM from each SWDM-multiplexed FBG sensor are used to mask the transmission power spectra. The mask excludes the data around each peak position with a width of double the FWHM. From this, reference power spectra are calculated by following algorithm 5. This algorithm, in essence, calculates a shared reference spectral profile that accounts for the non-flatness and a per-sample offset that accounts for changes in time. Then the refined reference is used to compute the refined reflection spectra in the same manner as done previously with the rough reference.

Algorithm 5 Power reference iterative estimation

Require: $x, y, n_{iteration}$
$M, N \leftarrow x.shape$
$x_{masked} \leftarrow mask(x, y)$ ▷ Mask spectra x from estimated wavelengths y.
$offset \leftarrow zeros(M)$
$profile \leftarrow zeros(N)$
for $j \in [1, \cdots, n_{iteration}]$ **do**
$x_{intp} \leftarrow interpolate(x_{masked}, axis = 1)$ ▷ Per spectral measurement interpolation.
$profile \leftarrow mean(x_{intp} - offset, axis = 0)$
$offset \leftarrow mean(profile - x_{masked}, axis = 1)$ ▷ Mean of non-masked elements.
end for
$reference \leftarrow profile - offset$
return $reference$

Appendix B

Frameworks

B.1 LEAP: Library for evolutionary algorithms in python

LEAP [103] is a general purpose Evolutionary Computation package with a strong priority on readability. It has features such as basic EA functional operators, distribution capabilities and visualization features. The basic workflow is based on the definition of a population of individuals that loop through a pipeline of operators that are applied sequentially.

The `Individual` class represents a solution by a *genome* and a *fitness*. It also maintains a reference to a `Problem` with which it will be evaluated and a reference to a `Decoder` which translates the *genome* into a *phenome* for fitness evaluation.

The `Decoder` class has a *decode* method that must be defined to compute the *phenome* from the *genome*. For our problem, the genome represents the spectral positions and the phenome the simulation for that spectral position. We define the decoder called `FBGDecoder` as

```
class FBGDecoder(Decoder):
    def decode(self, genome, *args, **kwargs):
        A_b = genome
        phenome = X(A_b, vars.λ, vars.A, vars.Δλ, vars.I,
                    vars.Δn_dc, simulation=vars.simulation)
        return phenome
```

The `Problem` class contains a *maximize* attribute that defines if the fitness must be maximized. It also requires the definition of the *evaluate* method which is used to calculate the *fitness* from the *phenome* together with the definition of *equivalent* and *worst_than* methods for *fitness* comparison. For our problem we define as a subclass of `ScalarProblem` (that implements standard scalar fitness comparison) the class `FBGProblem` that evaluates the fitness as the MAE.

```
class FBGProblem(ScalarProblem):
    def __init__(self, x=None):
```

```
        super().__init__(maximize=False)
        self.x = x
    def set(self, x):
        self.x = x
    def evaluate(self, phenome, *args, **kwargs):
        return np.sum(np.abs(self.x - phenome))
```

With the components of the `Individual` fully defined we can create a population and loop through a pipeline of operators to iteratively yield new populations and arrive at an optimal solution to the problem. A simple example is illustrated in the following code

```
""" ... define population size, bounds of genes, number of
        generations and spectra """
problem = FBGProblem(x)
parents = Individual.create_population(
                        pop_size,
                        initialize=create_real_vector(bounds),
                        decoder=FBGDecoder(),
                        problem=problem
                        )
parents = ops.grouped_evaluate(parents, problem=problem)
count=0
while count < max_generations:
    offspring = pipeline(parents)
    parents = offspring
    count += 1
best_individual = max(offspring)
```

For example, the pipeline for GA Real whose pipeline consists in stochastic universal sampling, crossover, mutation and comparison with the parent population is defined as

```
    def pipeline(parents):
        return pipe(
                    parents,
                    ops.sus_selection(
                        offset='pop-min',
                        key=lambda x: -x.fitness
                        ),
                    ops.clone,
                    ops.uniform_crossover(p_swap=p_crossover),
                    mutate_gaussian(std=std,
                                    expected_num_mutations=1,
                                    hard_bounds=bounds),
                    ops.pool(size=pop_size),
```

```
                    ops.grouped_evaluate(problem=problem),
                    ops.truncation_selection(size=pop_size,
                                             parents=parents)
                    )
```

B.2 Pytorch

Pytorch is an open source ML framework tailored for the development of DL applications. Its two high level features are its Tensor computation with GPU capability and its automatic differentiation system. It also contains all the required elements for building NNs, such as layers, loss functions and optimizers, together with additional utilities such as dataloaders. There are other more advanced features, such as distributed training available for the experienced practitioner.

The syntax of Pytorch follows closely the one of Numpy, the de facto tensor computation library of python. Pytorch implements an imperative programming model where each line of code is executed immediately as it is written, in contrast to an asynchronous model as seen in other frameworks such as Tensorflow. This approach, commonly known as *eager execution*, allows for better debugging and faster development as users can directly inspect and manipulate intermediate results at runtime.

The automatic differentiation system of Pytorch is called *autograd*. It fundamentally keeps a record of tensors and all executed operations in a directed acyclic graph (DAG) consisting of Function objects. In the DAG, the leaves are the input tensors and the roots are the output tensors. By tracing from roots to leaves through the graph the gradients can be obtained using the chain rule.

A simple example with the basic tensor computation and autograd capabilities is displayed in the following block of code.

```
# initialize tensor
input = torch.tensor([1, 1])
# Move to GPU if available
device = "cuda" if torch.cuda.is_available() else "cpu"
tensor = tensor.to(device)
# define random weight parameter
weight = torch.randn(2, 3, requires_grad=True, device=device)
# compute output as matrix multiplication of input and weight
output = torch.matmul(input, weight)
# compute simple loss as sum of output
loss = torch.sum(output)
# call backward pass to compute gradients
loss.backward()
```

```
# update weight by a single step of gradient descent
weight -= 0.01*weight.grad
```

The `torch.nn` namespace contains all the elements required to build a NN. Each building block in this namespace is an `nn.Module`. We can define an arbitrary model by composing other modules. The module must define its parameters or submodules in the `__init__` method, and define the computation of the output in the `forward` method. An example for a simple FCNN with 3 inputs, a hidden layer of size 2 with ReLU activation function and a two class output with a softmax is shown in the following block of code.

```
from torch import nn
import torch.nn.functional as F

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        linear_hidden = nn.Linear(in_features=3,
                                  out_features=2)
        linnear_output = nn.Linear(in_features=2,
                                   out_features=2)
    def forward(self, x):
        h = F.relu(linear_hidden(x))
        y = F.softmax(linear_output(h))
        return y
```

The parameters of the model can be optimized through gradients descent. A simple training loop, which corresponds to a single epoch of training, is shown in the following code block.

```
""" ... define hyperparameters and load training data """
train_dataloader = DataLoader(training_data, batch_size)
model = NeuralNetwork()
loss_function = F.cross_entropy
optimizer = torch.optim.SGD(model.parameters(),
                            learning_rate)
for X, y in train_dataloader:
    prediction = model(X)
    loss = loss_function(prediction, y)
    optimizer.zero_grad() # set gradients to zero
    loss.backward() # calculate gradients
    optimizer.step() # update parameters
```

B.3 Pytorch Lightning

Pytorch Lightning is a high-level DL framework built on top of Pytorch. The essence of this package is to reduce the boilerplate code associated with training without compromising the flexibility of Pytorch. It allows to disentangle the design of a model and the training, together with the possibility of adding complex training functionalities in a seamless way.

In Lightning the `LightningModule`, a subclass of `nn.Module` is the same in essence but also adds more functionalities, such as defining the configuration of the optimizer, the training loop and testing loop. In this way the module we defined in the previous section can be defined together with all necessary components for training inside a `LightningModule`, this is shown in the following block of code.

```
import pytorch_lightning as pl
import torch.nn.functional as F

class NeuralNetwork(pl.LightningModule):
    def __init__(self, learning_rate):
        super().__init__()
        self.learning_rate = learning_rate
        linear_hidden = nn.Linear(in_features=3,
                                  out_features=2)
        linnear_output = nn.Linear(in_features=2,
                                   out_features=2)
    def forward(self, x):
        h = F.relu(linear_hidden(x))
        y = F.softmax(linear_output(h))
        return y
    def configure_optimizers(self):
        optimizer = torch.optim.SGD(self.parameters(),
                                    self.learning_rate)
        return optimizer
    def training_loop(self, train_batch, batch_idx):
        x, y = train_batch
        prediction = model(x)
        loss = F.cross_entropy(prediction, y)
        self.log('train_loss', loss)
        return loss
    def validation_step(self, val_batch, batch_idx):
        x, y = val_batch
        prediction = model(x)
        loss = F.cross_entropy(prediction, y)
        self.log('val_loss', loss)
        return loss
```

Then the model can be trained in a GPU if available in a couple lines of code as

```
""" ... define hyperparameters and load data """
train_dataloader = DataLoader(train_data, batch_size)
val_dataloader = DataLoader(val_data, batch_size)
model = NeuralNetwork(learning_rate, batch_size)
trainer = pl.Trainer(max_epochs=max_epochs)
trainer.fit(model, train_dataloader, val_dataloader,
            accelerator="auto")
```

Other functionalities such as logging, checkpointing, gradient clipping, distributed training and mixed precision can be easily added just by passing appropriate arguments to the `Trainer` instance.

B.4 Optuna

Optuna [104] is an hyperparameter optimization framework, tailored to be used for ML. It has a *define-by-run* API that allows it to dynamically construct hyperparameter search spaces. Some of its features are the implementation of multiple state-of-the-art hyperparmeter optimization algorithms, integrated parallelization to run multiple trials[23] in different workers and visualization utilities.

The basic workflow is to define a `objective` function that takes as argument a *trial* object and returns a metric we want to minimize. Inside the `objective` the *trial* suggestion methods are used to sample the required hyperparameters. An *study* object is created to hold the optimization trials and the optimization is made by calling its *optimize* method that takes as argument the defined *objective* and the number of trials to run. After running the optimization, the best parameter set can be obtained from its *best_param* attribute. A simple example is shown in the following block of code, where we suggest a float $x \in [-10, 10]$ that minimizes the function $f(x) = (x - 2)^2$.

```
import optuna

def objective(trial):
    x = trial.suggest_float('x', -10, 10)
    return (x - 2) ** 2

study = optuna.create_study()
study.optimize(objective, n_trials=100)

study.best_params  # E.g. {'x': 2.002108042}
```

[23] trial refers to a single evaluation of the optimized function with an instance of hyperparameters

As we stated before we can dynamically modify the hyperparameter search space. An example of the usefulness of this can be seen in finding an optimal number of layers for an NN as well as the number of units of each layer, where we can see that the number of hyperparameters grows when increasing the number of layers. An example of this using pytorch to create an NN with a maximum of 3 layers and a variable number of units per layer between 4 and 128 is shown in the following block of code.

```
import torch
import optuna

def objective(trial):
    n_layers = trial.suggest_int('n_layers', 1, 3)
    layers = []

    in_features = 28 * 28
    for i in range(n_layers):
        out_features = trial.suggest_int(f'n_units_l{i}',
                                         4, 128)
        layers.append(torch.nn.Linear(in_features,
                                      out_features))
        layers.append(torch.nn.ReLU())
        in_features = out_features
    layers.append(torch.nn.Linear(in_features, 10))
    layers.append(torch.nn.LogSoftmax(dim=1))
    model = torch.nn.Sequential(*layers))
    # ... train model and calculate a performance metric
    return performance

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

Appendix C

Temperature Controller

As described in section 5.4.1, we require a high precision temperature controller. For this we design a temperature controller based on a Peltier, controlled with a temperature sensor as reference and a Proportional Integrative Derivative (PID) controller that controls the power and polarity delivered through an H-Bridge.

A Peltier is a solid-state device that transfers heat from one face to the other when electrical power is applied. If the polarity is inverted the heat transfer direction is inverted. This allows a Peltier element to be used to either cool or heat by inverting the polarity. By attaching a Peltier over a heatsink with a fan, we set one face as reference, to be set close to ambient temperature. Then with the Peltier we can either raise or lower the temperature of the opposite face.

In particular, we use the cooler of a CPU, as shown in figure C.1. Thermal paste is added in the interface between the heatsink and the Peltier. A temperature sensor is positioned on the top face of the Peltier with the FBGs positioned next to it. Thermal paste is added to the interface between the sensors and the Peltier, and these are fixed to the Peltier by use of insulating tape.

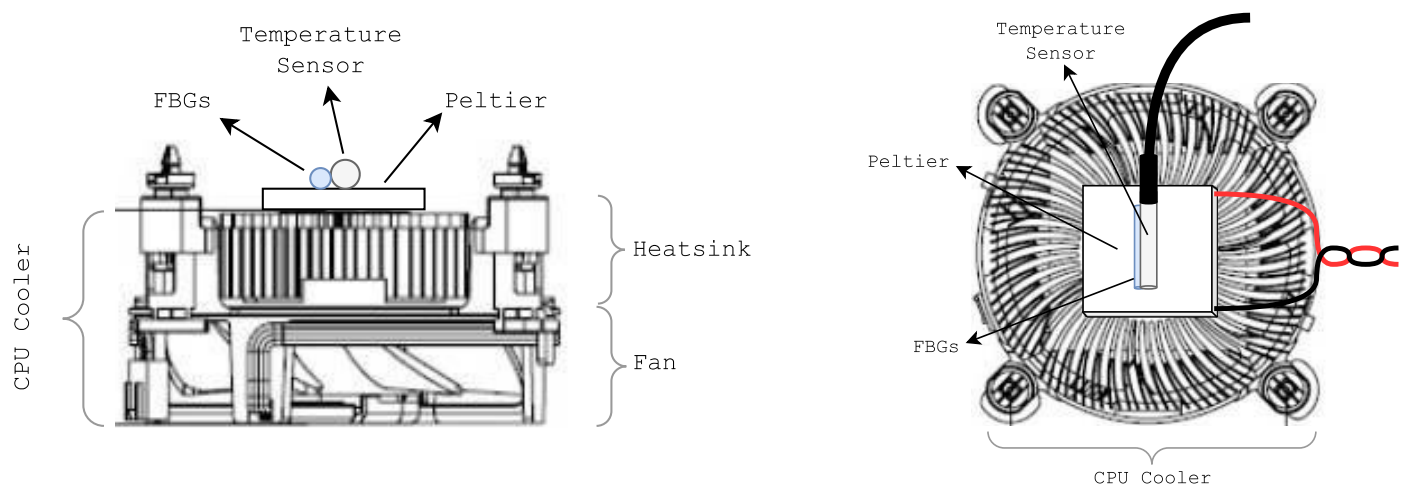


Figure C.1: Temperature controller side view (left). Temperature controller top view (right).

The control logic is performed by a microcontroller (μC) . The information flow logic is displayed in figure C.2. The polarity is set by comparing the target temperature and the ambient temperature [24] setting the peltier as heater or coooler respectively. The electrical power is controlled by a PWM signal of the microcontroller delivered to the H-Bridge. The PID updates the PWM values by looking at the difference between target temperatures and sensor temperatures. Separate PID parameters are set for cooling and heating.

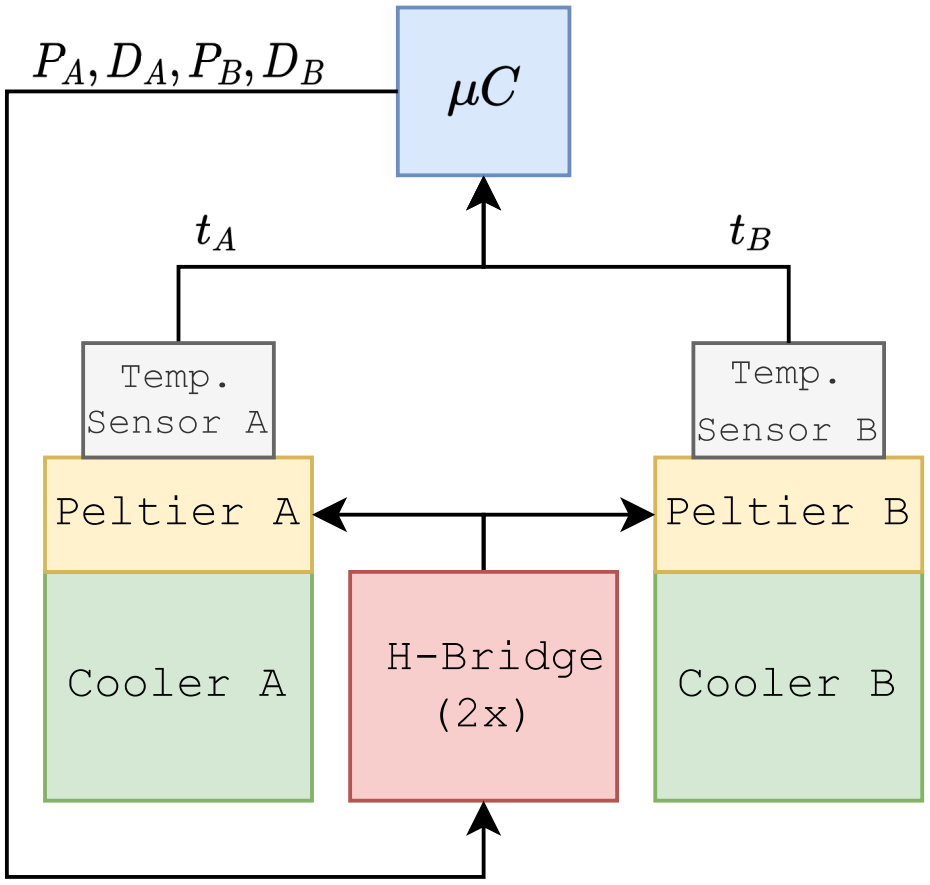


Figure C.2: Information flow diagram of controller. P_X and D_X represent the power ratio and the polarization for the peltier X, respectively. t_X is the temperature reading of the temperature sensor X.

The used components are described in table C.1, and an schematic of circuit is displayed in figure C.3.

Table C.1: Temperature controller components.

Component	Model	Characteristics
Temperature Sensor	DS18B20	$\delta t = 0.06$, $\Delta T = 750[ms]$
H-Bridge	L298N	$I_{max} = 2A$ (Over 2 channels)
Microcontroller	Arduino Uno R3	
Peltier	TEC1-12706	$V_{nom} = 12V$, $I_{max} = 6.4A$

[24] The sensors values at the start can be used as ambient temperature reference, although this can change with time, which is problematic.

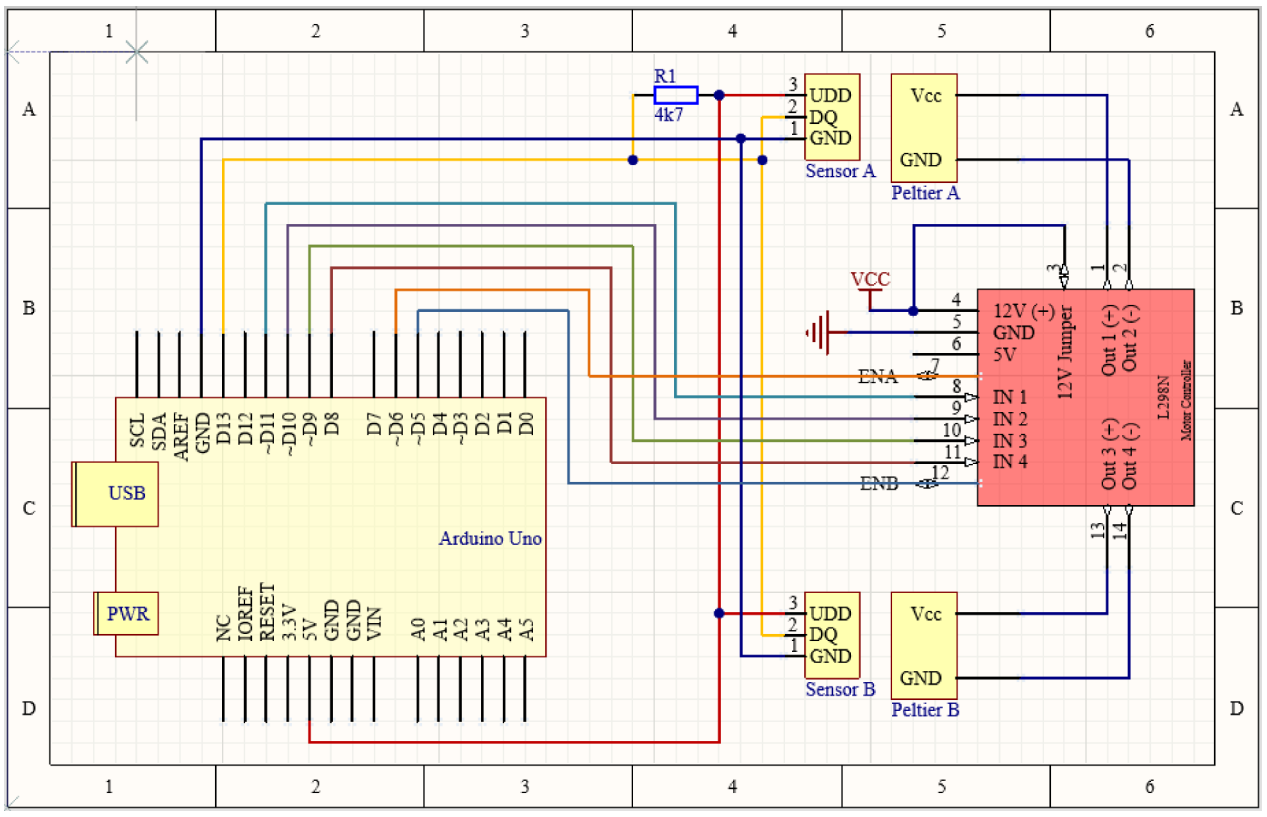


Figure C.3: Controller circuit schematic.