POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Deep Learning Methods for Inverse Supercontinuum Generation

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Andrea Corsini

Supervisor: Prof. Giacomo Boracchi

Advisors: M.Sc. Diego Stucchi, Prof. Alessandro Foi (Tampere University)

Academic year: 2020–2021

1. Introduction

Ultrafast optics studies the propagation of ultrashort pulses of laser light into strongly nonlinear media, e.g. optical fibre. This propagation results in the generation of broadband light known as the supercontinuum (SC) spectrum [2]. Generated spectra depend nonlinearly on parameters describing the laser light pulses, as well as fibre characteristics. Scientists necessitate SC generation in numerous scenarios, e.g. spectroscopy, optical tomography, meteorology. Depending on the scenario, the scientist may require a spectrum with certain characteristics. Therefore, generating SC spectra as close as possible to the required spectrum is an important task. However, knowing the correct parameters to generate the requested SC spectra is a complicated inverse problem because it would demand running continuously numerical simulations of the SC generation dynamic system, namely the generalized nonlinear Schrödinger equation (GNLSE).

As in other inverse problems of parameter estimation, deep learning techniques represent a possible solution. Neural networks can indeed learn from vast dataset generated by simulating the dynamic system. Such dataset consists of parameters-object pairs collected from the simulation forward pass, e.g. parameters-spectra in our specific case. These pairs can be leveraged to train deep learning models that learn to estimate the parameters.

In this thesis, we also propose two neural networks (NNs), fully-connected (FC) and convolutional (CNN), to solve the inverse problem of SC generation, i.e. estimating laser light pulses parameters generating a target SC spectrum.

Typically, the loss functions used to train neural networks to solve inverse problems penalize parameter errors, instead of errors in the forward domain. In nontrivial inverse problems, the penalization on the forward domain is impossible due to computational-demanding or unknown forward passes. To the best of our knowledge, we are the first to propose a surrogate weighted loss function that approximate the loss in the forward domain, i.e. the spectra.

We train the proposed networks by optimizing our weighted loss function. The method that computes the weighted loss is general, and it can be potentially applied to other parameter estimation inverse problems, regardless of the SC generation scenario. Compared to previous works [1] based on Genetic Algorithms, the neural networks have fast convergence and can predict the laser-pulse parameters for new spectra taking a few seconds on CPU.

We show that the weighted loss function is

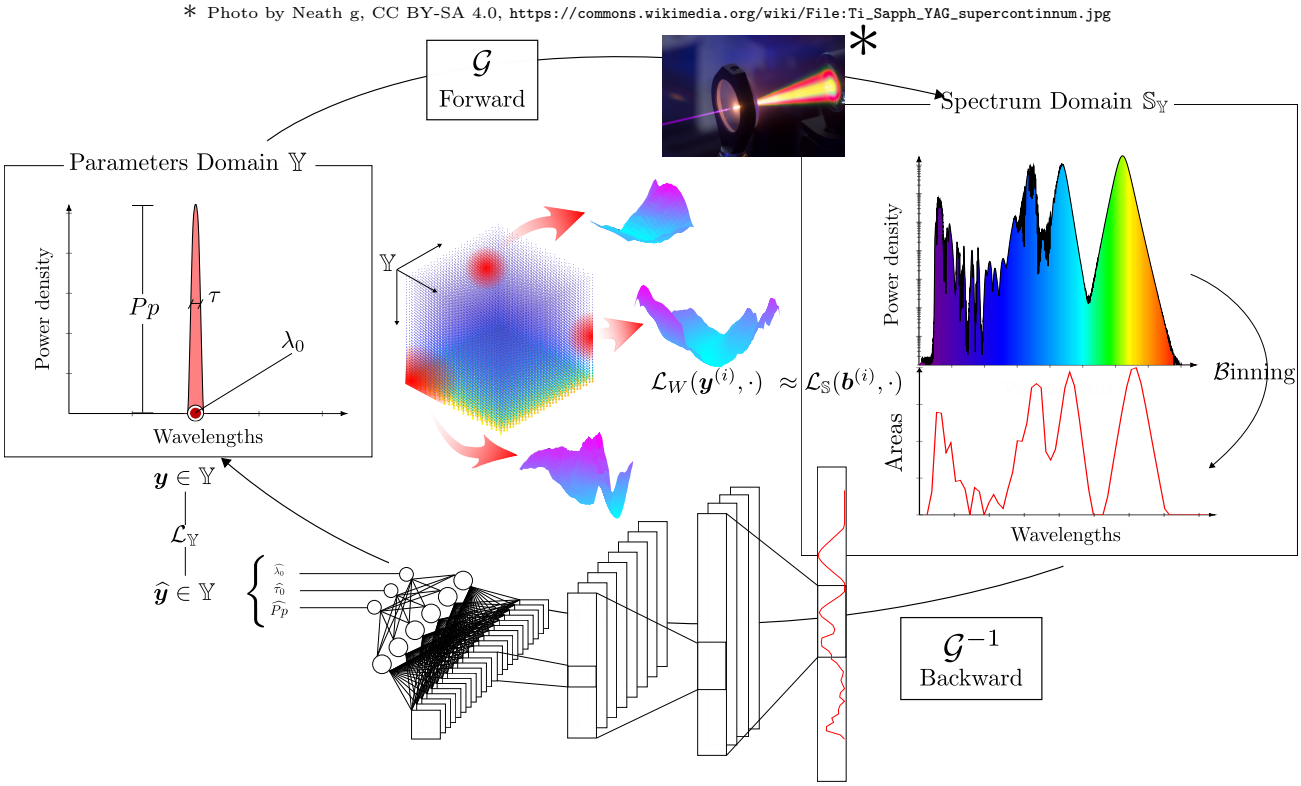


Figure 1: Overview of the problem and solution. In the considered inverse problem, the SC generation represents the forward pass from parameters to spectra. Fully-connected and convolutional networks learn the inverse process. From the parameters-spectrum pairs, we compute a weighted loss function $\mathcal{L}_W$ as nonparametric local polynomial approximation (LPA) of the spectrum loss $\mathcal{L}_\mathbb{S}$. We then emply the weighted loss function $\mathcal{L}_W$ to train the networks.

more beneficial than the conventional isotropic loss defined on the parameter space. Moreover, the NNs outperform naive methods such as the 1-nearest neighbour. Finally, we investigate the network results on non-generated spectra, namely Gaussian spectra and absorption spectra. The networks can predict laser parameters to some degree, and the weighted loss function helps move toward more accurate predictions.

Part of this thesis is founded upon our publication in [6].

2. Problem formulation

We define an arbitrary spectrum $\boldsymbol{s} \in \mathbb{S} \subset L^1(\mathbb{R})$ as a power density function of the wavelengths. Given the laser pump parameters $\boldsymbol{y} \in \mathbb{Y} \subset \mathbb{R}^P$, we can obtain supercontinuum spectra with the supercontinnum generator process $\mathcal{G}$ indicated as

$$\mathcal{G}(\boldsymbol{y}) : \mathbb{Y} \to \mathbb{S}_\mathbb{Y} \subset \mathbb{S}, \tag{1}$$

where $\mathbb{S}_\mathbb{Y}$ is the set of SC spectra that can be generated from $\mathbb{Y}$.

Our goal consists of learning the inverse of $\mathcal{G}$ by training a machine learning model $\mathcal{M} : \mathbb{S} \to \mathbb{Y}$ such that

$$\mathcal{M}(\boldsymbol{s}) = \underset{\boldsymbol{y}\in\mathbb{Y}}{\arg\min} \{d\left[\mathcal{G}(\boldsymbol{y}), \boldsymbol{s}\right]\}, \tag{2}$$

where $d : \mathbb{S}\times\mathbb{S} \to \mathbb{R}^+$ denotes a distance function over the spectrum domain. The distance function should consider main spectral features, while ignoring noise-like components, like those visible in the spectrum before binning of Figure 1.

A numerical simulation of the GNLSE implements the generator process $\mathcal{G}$. Hence, we adopt a discrete formulation of arbitrary spectra as vectors $\boldsymbol{\sigma} \in \mathbb{R}^N$ over wavelengths $\boldsymbol{\lambda} \in \mathbb{R}^N$. Therefore, $\mathbb{S} \subset \mathbb{R}^N \times \mathbb{R}^N$, where $N = 4096$ and is fixed by the simulator implementation.

The simulation is too expensive to embed in every iteration of model training loops. However, we can still execute it to generate a dataset TR of K spectrum-parameters pairs since this is a

one-time task:

$$TR = \{(\boldsymbol{s}^{(k)}, \boldsymbol{y}^{(k)}) \in \mathbb{S}_{\mathbb{Y}} \times \mathbb{Y} : \boldsymbol{s}^{(k)} = \mathcal{G}(\boldsymbol{y}^{(k)})\}_{k=1}^{K}, \tag{3}$$

We exclusively consider $P = 3$ laser-light pulse parameters, i.e. pulse wavelength λ_0, duration τ, and power peak Pp. We kept constant everything else, such as the fibre characteristics.

3. Proposed solution

We illustrate the proposed solution in Figure 1. Firstly, we preprocess the spectra to areas, reducing their dimensionality via Binning. Secondly, we feed the preprocessed spectra to the model $\mathcal{M}$. The model is trained to predict the parameters that generated the spectra in the forward pass. In this way, $\mathcal{M}$ can learn the inverse process $\mathcal{G}^{-1}$. Then, from the predictions we can assess the parameter error via parameters loss $\mathcal{L}_{\mathbb{Y}}$:

$$\mathcal{L}_{\mathbb{Y}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{P}\sum_{j=1}^{P}\|y_i - \widehat{y_i}\|. \tag{4}$$

Finally, the model predictions $\widehat{\boldsymbol{y}}$ can undergo the forward pass to generate the spectra $\widehat{\boldsymbol{s}}$. Hence, the performance can also be assessed by the spectrum loss $\mathcal{L}_{\mathbb{S}}$:

$$\mathcal{L}_{\mathbb{S}}(\boldsymbol{b}, \widehat{\boldsymbol{b}}) = \frac{1}{R}\sum_{j=1}^{R}\|b_j - \widehat{b}_j\|, \tag{5}$$

where $\boldsymbol{b}, \widehat{\boldsymbol{b}} \in \mathbb{R}^{60}$ denotes binned areas, i.e. preprocessed spectra.

Given the problem formulation in Section 2, we should optimize the model $\mathcal{M}$ with the spectrum loss $\mathcal{L}_{\mathbb{S}}$. However, this is impossible, because it would require to run the generator at every backpropagation step. Therefore, we optimize both models with our weighted loss function $\mathcal{L}_W$, i.e. the anisotropic surrogate loss over the parameter space that approximates the ideal spectral loss $\mathcal{L}_{\mathbb{S}}$.

3.1. Preprocessing

Firstly, we preprocess the dataset TR, by applying integral binning to every spectrum $(\boldsymbol{\lambda}, \boldsymbol{\sigma})$. Integral binning $\mathcal{B}$ is an operator that transforms spectra into n aggregated bins, namely binned areas $\boldsymbol{a}$, by integrating over a particular interval I. The binning aims at reducing the dimensionality $N = 4096$ of the spectrum power densities and wavelengths to $\mathbb{R}^n$. The lower dimensionality is particularly convenient for the FC model, in which the number of network trainable parameters would explode otherwise. Moreover, performing the binning over the same interval for each spectrum allows us to directly compare the preprocessed spectra on a reference wavelength grid.

Secondly, we convert the binned areas to dB values and, in some cases, we normalize the areas within the clipped interval of $[-40dB, 0]$. The normalization is applied only to the FC inputs and the arguments of $\mathcal{L}_{\mathbb{S}}$, but not to the CNN inputs.

3.2. Weighted loss function

We define the weighted loss function $\mathcal{L}_W : \mathbb{Y}^2 \to \mathbb{R}^+$ as a nonparametric local polynomial approximation (LPA) of a given spectral loss $\mathcal{L}_{\mathbb{S}} : \mathcal{B}(\mathbb{S})^2 \to \mathbb{R}^+$ over the binned areas. In particular, we compute a weighted loss for any i-th trainset pair $(\boldsymbol{s}^{(i)}, \boldsymbol{y}^{(i)})$ as

$$\mathcal{L}_W(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}) = \frac{1}{2}\sqrt{\boldsymbol{w}^{(i)\top} \cdot (\boldsymbol{y}^{(i)} - \widehat{\boldsymbol{y}})^2}, \tag{6}$$

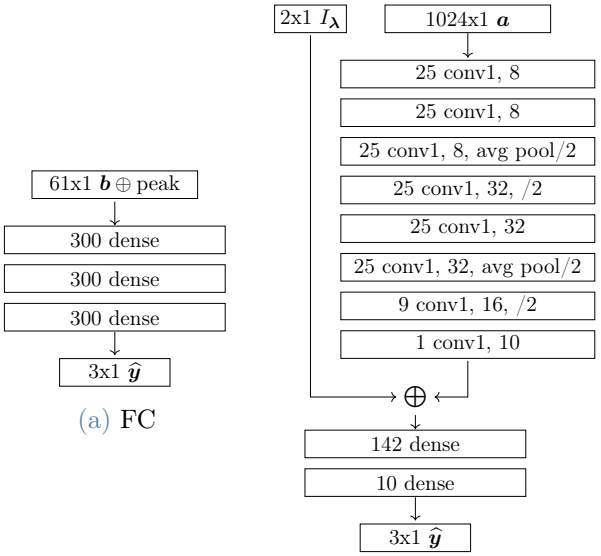
where $\boldsymbol{w}^{(i)} \in \mathbb{R}^P$ are the weights relative to the neighbour centred on the i-th trainset pair $(\boldsymbol{s}^{(i)}, \boldsymbol{y}^{(i)})$, $\widehat{\boldsymbol{y}} = \mathcal{M}(\boldsymbol{s}^{(i)})$ denotes the model prediction. Our method computes the weights $\boldsymbol{w}^{(i)}$ as

$$\boldsymbol{w}^{(i)} = \arg\min_{\boldsymbol{w}} \left\{ \sum_{j=1}^{K} c_j^{(i)} \cdot \right.$$
$$\left. \cdot \left[\mathcal{L}_{\mathbb{S}}^2\left(\boldsymbol{b}^{(i)}, \boldsymbol{b}^{(j)}\right) - \mathcal{L}_W^2\left(\boldsymbol{y}^{(i)}, \boldsymbol{y}^{(j)}\right)\right]^2 \right\}, \tag{7}$$

where $c_j^{(i)}$ are the locality coefficients of a P-dimensional isotropic Gaussian window with a fixed spread α, i.e. $c_j^{(i)} = \exp\{-\alpha\|\boldsymbol{y}^{(i)}, \boldsymbol{y}^{(j)}\|^2\}$. We apply the Dual Frame technique [3] to minimize Equation (7).

The ideal loss $\mathcal{L}_{\mathbb{S}}$ can denote any function of the areas, depending on the problem. In the case of SC spectra, we mainly consider the mean absolute error (MAE) from Equation 5. With non-SC-generated spectra, i.e. $\boldsymbol{s} \in \mathbb{S}$ but $\notin \mathbb{S}_{\mathbb{Y}}$, other spectrum loss functions are more helpful. For instance, we can approximate the intersection-over-union spectrum loss:

$$\mathcal{L}_W^{IOU} \to \mathcal{L}_{\mathbb{S}}^{IOU}(\boldsymbol{b}, \widehat{\boldsymbol{b}}) = \frac{\sum_j^R \min(b_j, \widehat{b}_j)}{\sum_j^R \max(b, \widehat{b}_j)}, \tag{8}$$



(a) FC (b) CNN

Figure 2: Neural network architectures. Blocks named *dense* and *conv1* denotes fully-connected and 1D convolutional layers, respectively. Every layer has the ReLU activation, except for the output ones that have linear activations.

where $\mathcal{L}_W^{IOU}$ indicates the weighted loss function computed over $\mathcal{L}_{\mathbb{S}}^{IOU}$. This loss function considers the shared energy between the two areas, rather than the shared peaks in the plain MAE of Equation 5.

3.3. Models

We propose two models, i.e. a fully-connected neural network (FC), and a convolutional neural network (CNN). Figure 2 illustrates their architecture.

The FC model receives in input binned normalized areas $\boldsymbol{b} \in \mathbb{R}^{60}$. The binned areas have been integrated on a common predefined grid I_C. Moreover, the peak value of each binned area is stacked to the input feature vector. On the other hand, the CNN model integrates each input $\boldsymbol{s} = (\boldsymbol{\sigma}, \boldsymbol{\lambda})$ in 1024 bins over the grid $I_\lambda = (\min\{\boldsymbol{\lambda}\}, \max\{\boldsymbol{\lambda}\})$, without performing the normalization. However, the integration grid extremes are passed to the network to help the localization of the spectrum in the wavelengths.

Additionally, we train each model $\mathcal{M}$ independently for $T = 10$ times with different trainable parameter initialisations. Eventually, we create a model ensemble $\overline{\mathcal{M}}$, where each prediction is the average of the predictions coming from the T models:

$$\widehat{\boldsymbol{y}} = \overline{\mathcal{M}}(\boldsymbol{s}) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{M}_t(\boldsymbol{s}). \tag{9}$$

This ensembling helps reducing the over-fitting and makes the evaluation more steady.

4. Findings

In the following experiments, we assess the performance of the neural networks. Firstly, we evaluates the neural networks on the testing set constituted by SC-generated spectra. Secondly, we investigate the generalization of NNs by predicting the parameters of non-SC-generated spectra, namely Gaussian and absorption spectra.

4.1. Testing set

To asses our methods, we split the dataset into F disjoint folds, and train an ensemble model $\overline{\mathcal{M}}_i$ on fold f_i for each architecture, where $i = 1, \ldots, F$. Then, we test each model $\overline{\mathcal{M}_i}$ on the next fold $f_{(i \mod F)+1}$. We accumulate every metric over the test folds. Specifically, we compute the parameters error $E_{\mathbb{Y}}$ and the spectrum error $E_{\mathbb{S}}$ with loss functions $\mathcal{L}_{\mathbb{Y}}$ and $\mathcal{L}_{\mathbb{S}}$, respectively. Despite the error $E_{\mathbb{S}}$ requires invoking the generator on model predictions, this generation is computationally feasible since it is a one-time task, like the training set generation. To study the impact of the training set size over model performances, we repeat the experiment for multiple numbers of folds $F = 5, 10, 20$. Table 1 summarises the results.

First, we found that models trained optimizing the weighted loss function $\mathcal{L}_W$ achieve lower spectrum error than models trained to minimize $\mathcal{L}_{\mathbb{Y}}$. One exception to this rule is the CNN trained on 20 folds, where the spectrum error increases. We hypothesize that this is due to the scarcity of samples in this category. Indeed, we can observe that the spectrum error improvements $\Delta E_{\mathbb{S}}$ reduces when the samples per training set diminish, i.e. folds increase. Second, both models outperform the 1-nearest neighbour baseline (*1N*) defined as the parameters of the most similar spectrum in the training fold f, i.e. $1N(\boldsymbol{b}^\star) = \arg\min_{(\boldsymbol{b},\boldsymbol{y})\in f} \mathcal{L}_{\mathbb{S}}(\boldsymbol{b}, \boldsymbol{b}^\star)$. Third, despite the CNN having almost half of parameters of the FC, the CNN unsurprisingly achieves lower errors than the FC model. Among the reasons, we remark that the CNN applies a less

		$F=5$			$F=10$			$F=20$		
		$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$
1N		0.123	0.354	-	0.161	0.449	-	0.206	0.569	-
FC	$\mathcal{L}_{\mathbb{Y}}$	0.029	0.168	(-7.1%)	0.045	0.235	(-8.5%)	0.066	0.329	(-5.5%)
	$\mathcal{L}_W$	0.032	0.156		0.047	0.215		0.070	0.311	
CNN	$\mathcal{L}_{\mathbb{Y}}$	0.014	0.130	(-11.5%)	0.022	0.186	(-7.5%)	0.034	0.268	(+1.5%)
	$\mathcal{L}_W$	0.017	0.115		0.028	0.172		0.044	0.272	

Table 1: Results of model predictions over the testing generated spectra. Each row consider a different model, namely 1-nearest neighbour *1N*, fully-connected FC and convolutional neural network CNN. We trained the neural networks both with parameter loss $\mathcal{L}_{\mathbb{Y}}$ and weighted loss $\mathcal{L}_W$. We repeated each training-test procedure for the dataset split in $F = 15, 10, 20$ folds.

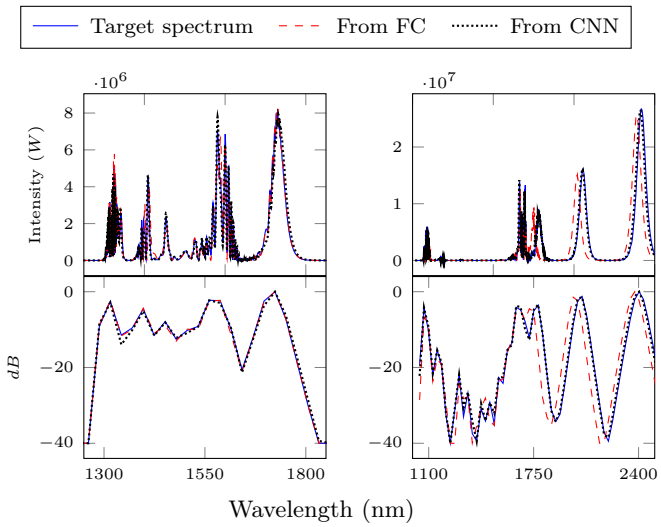


Figure 3: Examples of spectra (above) and relative areas (below) from testing set predictions. In each figure, we plot the original target spectrum and the spectra generated from the prediction of FC and CNN models. Both samples are randomly drawn from the testing set, respectively from the FC spectrum error $E_{\mathbb{S}}$ intervals of $[40, 60]$-percentiles and $[90, 100]$-percentiles.

coarse preprocessing than the FC since we expect the CNN to learn the feature extractor.

Finally, the spectra generated from the predictions of the models are very close to the target spectra. As Figure 3 highlights, this is true even for the testing samples with higher spectrum error.

4.2. Non-SC-generated spectra

Additionally, we examine the generalization of our method by testing on two examples of arbitrary spectrum groups, namely Gaussian and absorption spectra. On the one hand, the Gaussian spectra mimic user hand-drawn profiles, which are more imprecise than SC spectra. On the other hand, absorption spectra describe highly accurate wavelengths absorbed by substances. For these reasons, these tasks are ambitious since generating their spectra with $\mathcal{G}$ may be physically impossible.

In the case of Gaussian spectra, we test over a dataset constituted by fitting of Gaussian Mixture models on the testing set. We report two examples in Figure 4. We can see that the networks are able to predict decent laser parameters that lead to reasonably close spectra. Even in case of degenerate Gaussian areas (Figure 4 right), the generated spectra exhibit similar features to the target ones. Therefore, we stress the challenge further, by replacing the Gaussian Mixture fitting with sum of random Gaussian functions. In this case, the prediction quality is promising but requires improvements.

In the case of absorption spectra, the models can partially predict decent laser parameters $\widehat{\boldsymbol{y}}$ that generate close spectra $\widehat{\boldsymbol{s}}$ to some degree. Nevertheless, predictions are reasonable enough to concentrate the power densities of $\widehat{\boldsymbol{s}}$ around the target wavelengths of interest. In particular, we notice that models trained with $\mathcal{L}_W^{IOU}$ (8) tend to lead to more plausible profiles. Figure 5 highlights this effect by comparing the result of $\mathcal{L}_W^{IOU}$ and $\mathcal{L}_{\mathbb{Y}}$ on the Acetylene absorption spectrum. We can observe that the FC trained without weighted loss don't even generate the parameters for an envelope of the target spectrum.

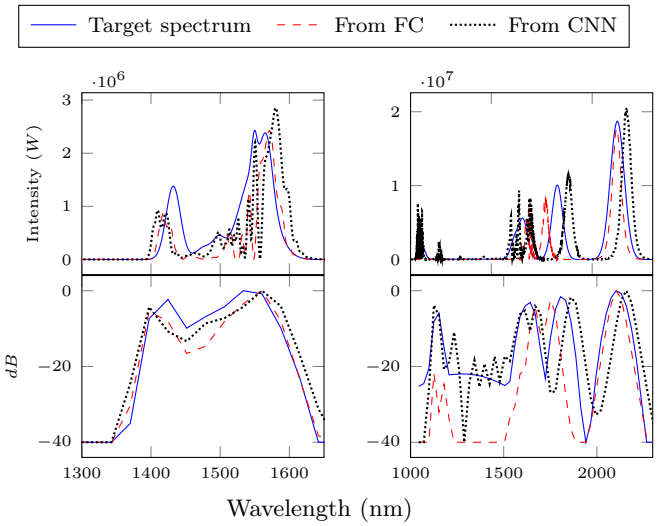


Figure 4: Examples of spectra (above) and related areas (below) generated from Gaussian dataset predictions. Similarly to Figure 3, the samples are drawn from the FC $E_{\mathbb{S}}$ percentile intervals $[26, 50]$ and $[76, 100]$.

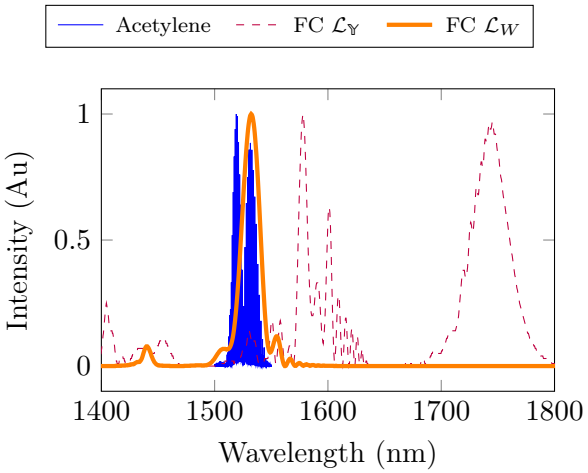


Figure 5: Spectra generated from model predictions on Acetylene. This figure compares predictions from a FC model trained with $\mathcal{L}_W^{IOU}$ (8) against the same model trained with the parameter loss $\mathcal{L}_{\mathbb{Y}}$ (4).

5. Conclusions

We propose a novel method to train neural networks for parameter estimation inverse problems leveraging a surrogate loss function for the forward domain. Researchers can potentially apply this method to other parameter estimation inverse problems where the forward pass is not available at training time. In our case, we employ this method to solve the inverse supercontinuum generation.

Future work will focus on multiple directions. We will extend our method by considering spatially adaptive approximations [4]. We believe that adaptive weighted schemes can improve the predictions on arbitrary spectra. Then, we will test the validity of our current method and its future extensions on other parameter estimation inverse problems, such as the estimation of airfoil shape parameters from wind velocity maps [5]. Regarding arbitrary spectra, we will investigate both changes to current models and brand-new architectures to improve the results.

References

[1] F. Arteaga-Sierra, C. Milián, I. Torres-Gómez, M. Torres-Cisneros, H. Plascencia-Mora, G. Moltó, and A. Ferrando. Optimization for maximum Raman frequency conversion in supercontinuum sources using genetic algorithms. *Revista Mexicana de Física*, 63.

[2] J. M. Dudley, G. Genty, and S. Coen. Supercontinuum generation in photonic crystal fiber. *Reviews of Modern Physics*, 78(4):1135, 2006.

[3] G. Farnebäck. A unified framework for bases, frames, subspace bases, and subspace frames. In *Proceedings of the 11th Scandinavian Conference on Image Analysis*, pages 341–349, 1999.

[4] A. Goldenshluger and A. Nemirovski. On spatially adaptive estimation of nonparametric regression. *Mathematical methods of Statistics*, 6(2):135–170, 1997.

[5] A. Schillaci, M. Quadrio, C. Pipolo, M. Restelli, and G. Boracchi. Inferring functional properties from fluid dynamics features. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4091–4098. IEEE, 2021.

[6] D. Stucchi, A. Corsini, G. Genty, G. Boracchi, and A. Foi. A weighted loss function to predict control parameters for supercontinuum generation via neural networks. In *2021 IEEE 31th International Workshop on Machine Learning for Signal Processing Proceeding (MLSP)*. IEEE, 11 2021.

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Deep Learning Methods for Inverse Supercontinuum Generation

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Andrea Corsini**

Student ID: 920008
Supervisor: Prof. Giacomo Boracchi
Advisors: M.Sc. Diego Stucchi (Politecnico Di Milano),
Prof. Alessandro Foi (Tampere University)
Academic Year: 2020-2021

Abstract

Supercontinuum generation creates broadband spectra from the propagation of short light pulses emitted by a laser pump into optical fibres. The characteristics of the pulses determine the profile of the generated spectrum. Therefore, scientists can choose the parameters to control the laser light pulses. However, determining the parameters of desired spectrum profiles is complicated, due to the nonlinearity of the process. It is also possible to simulate this process by forwarding the parameters to computationally-intensive numerical solvers. Therefore, machine learning models can leverage simulated samples to learn the inverse process. However, the training procedures can not invoke the costly simulation.

We propose two neural networks, fully-connected and convolutional, to estimate the parameters that generate spectra as close as possible to desired ones. We train the models via our proposed weighted loss function. The weighted loss function approximates the ideal loss over the spectra space. The method is generic and potentially transferable to other inverse problems that necessitate computational-expensive simulations. We show that the loss function improves the quality of the predictions, and it can help achieve spectra closer to the desired ones. We discuss issues in adopting the proposed solution to arbitrary spectra, i.e. not produced by the supercontinuum process.

Keywords: inverse problem, supercontinuum, loss function, deep learning

Sommario

La generazione supercontinua crea spettri a banda larga dalla propagazione di brevi impulsi di luce emessi dai laser in fibre ottiche. Le caratteristiche degli impulsi determinano la forma dello spettro generato. Di conseguenza, gli scienzati possono scegliere i parametri per modificare gli impulsi di luce del laser. Tuttavia, trovare i parametri dello spettro desiderato è complicato, data la nonlinearità del processo. Comunque, è possibile simulare questo processo fornendo i parametri a risolutori numerici dall'alto costo computazionale. Quindi, modelli di apprendimento automatico possono imparare il processo inverso dai dati simulati. Ciononostante, il processo di allenamento non potrebbe mai invocare le costose simulazioni al suo interno.

Proponiamo due reti neurali, interconnesse e convoluzionali, per stimare i parametri che generano spettri il più vicini ai desiderati. Alleniamo entrambi i modelli con una nuova funzione di costo pesata che proponiamo. La funzione di costo approssima quella ideale definita sullo spazio degli spettri. Il nostro metodo è generale e potenzialmente trasferibile ad altri problemi inversi, i quali simulatori sono computazionalmente costosi. Nella tesi, mostriamo che la nostra funzione di costo migliora la qualità delle predizioni. Inoltre, può aiutare a generare spettri che siano ancora più vicini ai desiderati. Discutiamo anche di problemi nell'applicare la soluzione a spettri arbitrari, cioè non prodotti dalla generazione supercontinua.

Parole chiave: problema inverso, supercontinuo, funzione di costo, apprendimento profondo

Preface

We worked on this research during my permanence at the Signal and Image Restoration group at Tampere University. We collaborated with the Ultrafast Photonics Group at the same university. Part of the thesis manuscript is based on the following co-authored publication:

> Diego Stucchi, Andrea Corsini, Goëry Genty, Giacomo Boracchi, and Alessandro Foi. A weighted loss function to predict control parameters for supercontinuum generation via neural networks. In *2021 IEEE 31th International Workshop on Machine Learning for Signal Processing Proceeding (MLSP)*. IEEE, 11 2021

Acknowledgements

Firstly, I want to thank my thesis supervisor Prof. Giacomo Boracchi for having introduced me to the group at Tampere University and allowing me to work on my Master's thesis in this environment.

My deep gratitude goes to my advisor from Politecnico di Milano M.Sc. Diego Stucchi. I will always be grateful to Diego for closely guiding my work during these months, discussing advancements, and answering my questions. I want to thank also my advisor from Tampere University, Prof. Alessandro Foi, that together with Giacomo, provided invaluable support to this work. All of them welcomed me into the research group, from which I had the chance to learn a lot. Moreover, I want to thank the *Academy of Finland* for supporting our research.

Special acknowledgement to the EIT Digital Master School, which admitted me in such a vibrant and exciting master's degree programme. In particular, I want to express my gratitude to Federico Schiepatti for constantly having helped us in each phase of the programme.

I am also thankful to all my close friends that have accompanied me during these years, starting from Lufan and the Politong group from the Shanghai days, until Fabio and all the others from Helsinki and EIT Digital. Thanks also to Vila for the constant support and presence.

Last but not least, I want to thank my parents and my sister. They continuously

encouraged me in all of my choices. Without their absolute love, all my studies would not have been possible.

Milano, December, 21, 2021

Andrea Corsini

Abbreviations and Acronyms

Au	Arbitrary unit
CNN	Convolutional Neural Network
DL	Deep Learning
FC	Fully Connected
GNLSE	Generalized Nonlinear Schrödinger Equation
GA	Genetic Algorithm
GMM	Gaussian Mixture Model
GPU	Graphics Processing Units
HPT	Hyperparameter Tuning
MAE	Mean Absolute Error
ML	Machine Learning
NN	Neural Network
ReLU	Rectified Linear Unit
SC	Supercontinuum
SRG	Sum of Random Gaussian Curves
SGD	Stochastic Gradient Descent
TFWHM	Temporal Full Width at Half Maximum

Notation

Main general symbols:

$\boldsymbol{x}$	A vector.
x_i	i-th element of vector $\boldsymbol{x}$.
$\boldsymbol{x}^{(n)}$	n-th vector of a collection, such as a set or dataset.
$\boldsymbol{A}$	A matrix.
A_{ij}	Element of matrix $\boldsymbol{A}$ at i-th row and j-th column.
$\boldsymbol{A}^\top$	The (conjugate) transpose of $\boldsymbol{A}$.
$\boldsymbol{A}^\dagger$	The pseudoinverse of $\boldsymbol{A}$.
$\widetilde{\boldsymbol{A}}$	The dual frame of $\boldsymbol{A}$.

Main specific symbols:

$\mathbb{Y}$	Set of laser parameters.
$\mathbb{S}$	Set of arbitrary spectra.
$\mathbb{S}_\mathbb{Y}$	Set of supercontinuum spectra generated from $PARAMS$
$\mathcal{L}_W$	Weighted loss function.
$\mathcal{L}_\mathbb{S}$	Weighted loss function.
$\mathcal{L}_\mathbb{Y}$	Weighted loss function.
$\mathcal{G}$	Supercontinuum generation, or generator.
$\mathcal{B}$	Integral binning operator.
$\boldsymbol{s} = (\boldsymbol{\lambda}, \boldsymbol{\sigma})$	Spectrum with power density $\boldsymbol{\sigma}$ across the wavelengths $\boldsymbol{\lambda}$.

Contents

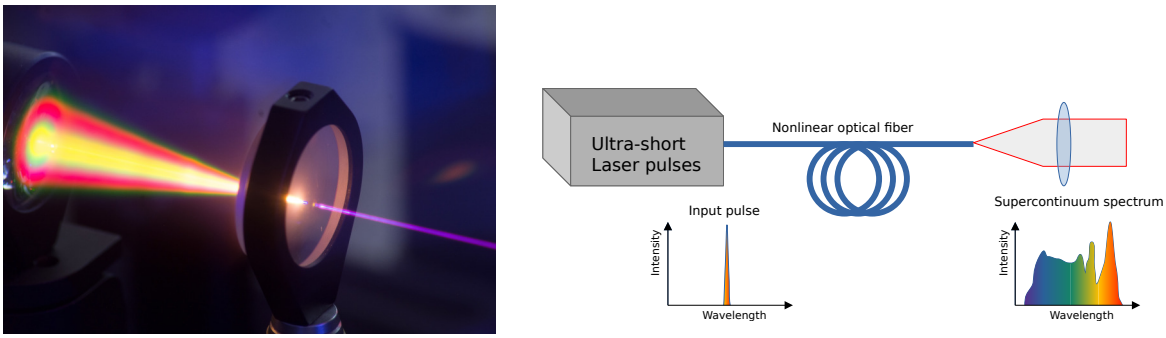
1 | Introduction

Supercontinuum (SC) lasers propagate high-power train pulses in a nonlinear optical fibre, producing broader spectra than classical monochromatic lasers [11]. Scientists employ the resulting SC spectra for numerous applications, e.g., high-precision metrology, optical coherence tomography (OCT), high-resolution imaging and remote sensing. Commonly, scientists can configure the laser parameters to obtain output SC spectra. Alternatively, they can predict the output spectrum for a given set of parameters by numerically simulating the generalized nonlinear Schrödinger equation (GNLSE) [2], i.e. the mathematical description of this system dynamics. Hence, it would be even more beneficial to invert the SC generation. In this way, scientists could know the necessary parameters to generate the target spectrum profile. For instance, OCT requires spectra in the near-infrared regions.

However, optimizing the SC generation output spectra is not a straightforward operation. This is due to the rich landscape of nonlinear dynamics that affects the SC generation. Determining the train pulses parameters to obtain the desired features is a hard and ill-posed inverse problem. Machine learning (ML) techniques have shown encouraging results in solving similar nonlinear inverse problem. In parameter estimation inverse problems, the differential equation that describes the system, e.g. the GNLSE for the SC generation, can be simulated for an arbitrary number of different parameter sets. Therefore, parameters and outputs can respectively become targets and inputs for the training of ML models. Hence, the ML model estimates the parameters given a generated output. In the case of gradient-descent-based models, such as neural networks (NNs), it is reasonable to propagate the error between the input and the object generated by the estimated parameters. In this way, the models minimize this difference.

However, in our example of inverse problems, the forward model of the SC generation is computationally-demanding. This also holds for all the other systems based on a nonlinear Schrödinger equation. Therefore, the training process can not directly embed the forward model to regenerate the outputs from predicted parameters. Hence, the loss over the outputs is unavailable, i.e. in our problem, the spectrum loss.

The goal of this thesis is to solve the SC inverse problem. Simultaneously, we want to minimize the distance between target spectra and generated spectra. Our proposition is two-fold. First, we propose two neural network architectures that predict the estimated



(a) Photo of SC generation in visible and near-IR wavelengths.[1]

(b) Schematised version of the SC generation.

Figure 1.1: Supercontinuum generation process.

parameters. Second, we offer a method to approximate the ideal spectrum loss via a weighted loss in the parameter domain. We employ this weighted loss function to train the neural networks.

We show that the NNs perform more accurately than 1-nearest-neighbour searches over the spectrum domain. We also show that the neural networks trained with the weighted loss achieve better performances in the spectrum error than those trained with isotropic losses on the parameters.

To further evaluate our solution, we consider two scenarios. First, Gaussian spectra, which resembles possible hand-drawn spectra that a user could sketch. Second, absorption spectra, adopted in science and industry to detect materials of interest. The generalization power of our solution is not perfect yet, but we aim at increasing it in future work. Nevertheless, the introduction of the weighted loss function visually improves the quality of the regenerated spectra.

1.1. Contributions

Our contributions include the design and training of two model architectures, fully-connected and convolutional neural networks, that solve the inverse problem of estimation of laser pulse parameters. Moreover, we also introduce a novel weighted loss function that optimizes neural networks for solving the inverse problem via approximation of the ideal loss function in the forward domain, i.e., the spectra domain. Remarkably, The adoption of our proposed weighted loss improves the estimation accuracy of both neural networks, by reducing the spectrum error between targets spectra and the ones generated

[1]From Neath g., Creative Commons BY-SA 4.0, `https://commons.wikimedia.org/wiki/File:Ti_Sapph_YAG_supercontinnum.jpg`.

from predictions. Nevertheless, this technique is not specific to the inverse supercontinuum generation, hence, it can potentially improve the accuracy in other inverse problems that require computationally expensive simulations.

From a practical point of view, our contributions also include an entire software framework for the comparison and fine-tuning of possible models. This framework takes care of several aspects, e.g. distributed hyperparameter tuning, SC generation, weighted loss computation, and metrics evaluation. Within this framework, we implement our proposed solution in Python and Matlab, respectively used for the neural networks and weighted loss approximation. Finally, we fit the proposed solution over a simulated dataset of parameters-spectrum pairs. This dataset has been generated by executing a Matlab solver provided by the Ultrafast Photonics Group of Tampere University.

1.2. Thesis structure

We organize the rest of this thesis as follows. Chapter 2 mathematically formulates the problem of inverse supercontinuum generation. Chapter 3 provides background knowledge to better understand the proposed solution, but also it reviews previous work both in the field of parameter estimation for supercontinuum generation and deep learning in inverse problems. Then, Chapter 4 illustrates the techniques designed to solve the SC problem. Chapter 5 briefly describes their implementation. Afterwards, Chapter 6 empirically evaluates the proposed solution. Evaluation continues in Chapter 7, that also discusses problems and limitations of our current solution. Finally, the thesis concludes in Chapter 8, where we introduce future research directions too.

2 | Problem formulation

Firstly, this chapter mathematically describes the spectra generation process and involved objects, i.e. spectra, SC generated spectra, and laser pump parameters. Then, given those definitions, we continue formalizing the inverse problem solved in this thesis.

2.1. Spectra generation

We define an arbitrary spectrum $\boldsymbol{s} \in \mathbb{S} \subset L^1(\mathbb{R})$ as a power density function over the wavelengths. We also define a supercontinuum spectrum as the result of the supercontinuum generation process

$$\mathcal{G} : \mathbb{Y} \to \mathbb{S}_{\mathbb{Y}} \subset \mathbb{S}, \tag{2.1}$$

where $\mathbb{Y} \subset \mathbb{R}^P$ denotes the set of parameters of the SC generation process, and $\mathbb{S}_{\mathbb{Y}} \subset \mathbb{S}$ represents the set of plausible spectra that the set $\mathbb{Y}$ can actually generate.

In this research, we exclusively consider $P = 3$ parameters that primarily influence the SC generation. Specifically we focus on the parameters that describes the train of laser pulses that propagate in the nonlinear fibre. The parameters $\boldsymbol{y}$ include the initial wavelength λ_0, the pulse duration τ, also known as temporal full width at half maximum (TFWHM), and the pulse power peak P_p. We maintain constant all the other SC generation parameters, e.g. the length of the nonlinear and its mode.

We consider only simulated SC spectra, i.e. the SC generation process $\mathcal{G}$ corresponds to a numerical integration of the GNLSE. In this way, it is possible to obtain a larger set $\mathbb{Y}$ by varying the SC generation parameters easily. Therefore, we adopt a discrete formulation of the arbitrary spectra, that can now be seen as discretized power density vectors $\boldsymbol{\sigma} \in \mathbb{R}^N$ associated to the wavelengths $\boldsymbol{\lambda} \in \mathbb{R}^N$, i.e. as the pair $\boldsymbol{s} = (\boldsymbol{\lambda}, \boldsymbol{\sigma}) \in \mathbb{S} = \mathbb{R}^N \times \mathbb{R}^N$, where $N = 4096$ is the dimensionality implemented in the adopted generator. Figure 2.1 reports three examples of spectra simulated by the generator.

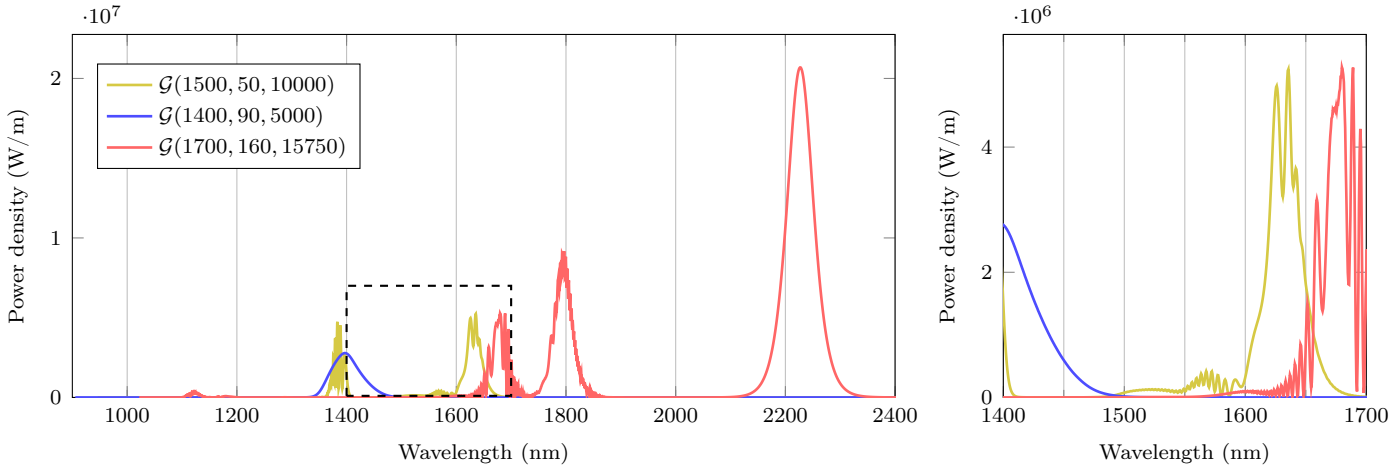


Figure 2.1: Example of output spectra from the generator. On the left, we show three possible generated SC spectra. On the right, we show the details of the three spectra in the wavelength interval $(1400, 1700)$ nm.

2.2. Inverse problem

This thesis focuses on solving the inverse problem of inferring the parameters $\boldsymbol{y} \in \mathbb{Y}$ that generate the desired spectrum $\boldsymbol{s} \in \mathbb{S}$. Since the GNLSE can not be inverted analytically, the resulting problem reduces to train an ML model $\mathcal{M} : \mathbb{S} \to \mathbb{Y}$ such that

$$\mathcal{M}(\boldsymbol{s}) = \arg\min_{\boldsymbol{y} \in \mathbb{Y}} \{d\left[\mathcal{G}(\boldsymbol{y}), \boldsymbol{s}\right]\}, \tag{2.2}$$

where the $d : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^+$ represents a generic distance function. We require the distance function to detect only the salient features of the spectra, possibly avoiding the impact of noise-like features and high frequencies illustrated in figure 2.1.

We assume that a training set TR of K spectrum-parameters samples is available to train the ML model, and is defined as

$$TR = \{(\boldsymbol{s}^{(k)}, \boldsymbol{y}^{(k)}) \in \mathbb{S}_{\mathbb{Y}} \times \mathbb{Y} : \boldsymbol{s}^{(k)} = \mathcal{G}(\boldsymbol{y}^{(k)})\}_{k=1}^{K}. \tag{2.3}$$

We remark that it is possible to invoke $\mathcal{G}$ during the generation of TR, as well as during the performance assessment of the model. However, its computationally-intensive requirements prevent the adoption during the training phase.

3 | Backgrounds

This chapter aims at providing the reader with sufficient information to familiarise with the methods behind the problem and solution of this thesis work. Section 3.1 presents the required deep learning fundamentals. In particular, we focus on fully-connected neural networks, convolutional networks, and training procedures via cost functions. Then, Section 3.2 reviews previous works in inverse Supercontinuum generation and deep learning applied to inverse problems. Finally, Section 3.3 explains the algebra behind local polynomial approximation.

3.1. Deep learning

Machine learning (ML) methods learn to solve several problems with data-oriented techniques. These problems usually comprises supervised, unsupervised and reinforced learning. Specific examples include regression, classification, clustering, taking actions in an environment. Typically, classical ML techniques require feature engineering processes, i.e. human experts define new features to be computed from the dataset and fed to ML algorithm. For example, a medical doctor provides a diagnosis from the patient symptomatology based on observed characteristics, like fever presence. Similarly, an ML algorithm, e.g. logistic regression, learns the correlation between a set of predefined features and the target decision. However, classical ML algorithms rely heavily on data representation. They cannot learn to perform a particular task from heterogeneous features like pixels in images, words in corpora.

Deep learning (DL) is a sophisticated ML technique that solves this problem. DL learns the representation from input data, as well as the mapping to the outputs. This is particularly useful when it is hard to understand how to codify features from data. For instance, in multi-class object classification tasks on images, it is hard to produce reliable features for each class. In this thesis, we solve regression tasks over signals. Besides some preprocessing that we could apply, it is not possible to engineer features further.

DL methods mimic the structure and functioning of the human brain. Like in the human brain, DL methods define structures called artificial neural networks (ANN). ANNs consist of graphs of neurons. Each neuron holds a value, namely, activation, computed by

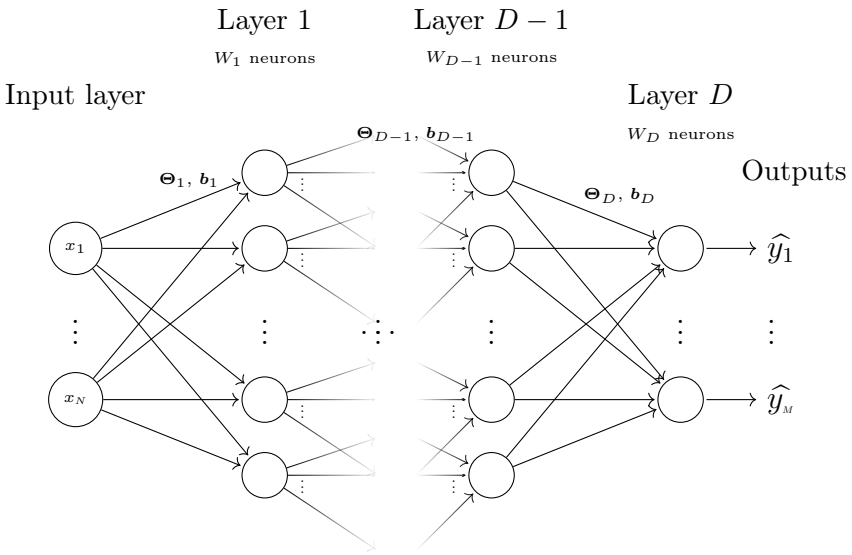


Figure 3.1: Generic fully-connected neural network representation with D hidden layers and N inputs and M outputs.

its input. The activation is then passed as input to other neurons.

Deep neural networks have proven to outperform traditional solutions in several fields, from image and computer vision to natural language processing. This thesis adopted two solutions based on fully-connected (FC) neural networks and convolutional neural networks (CNN), respectively described in sections 3.1.1 and 3.1.2.

3.1.1. Fully connected neural networks

Although the community often perceives DL as a modern technique, its lineage [33] has deeper roots than modern personal computers. Fully connected neural networks were among the firstly-introduced architectures.

Fully connected neural networks, also known as multi-layer perceptrons (MLP) and feedforward artificial neural networks, dispose neurons into a particular kind of direct acyclic graphs. These graphs partition the neurons into several groups called layers. The number of layers D determines the depth of the network, and the number of neurons in the i-th layer W_i determines the layer width. As Figure 3.1 shows, each neuron in the i-th layer receives the values held in the previous one, the $(i-1)$-th layer, for $i = 1, \ldots, D$, where the 0-th layer simply holds the input data.

Mathematically, we can represent FC layers as a simple composition of functions, such as

$$\widehat{\boldsymbol{y}} = (f_D \circ f_{D-1} \circ \cdots \circ f_2 \circ f_1)(\boldsymbol{x}), \tag{3.1}$$

$$f_i(\boldsymbol{x};\, \boldsymbol{\Theta}_i, \boldsymbol{b}_i) = a_i(\boldsymbol{\Theta}_i \boldsymbol{x} + \boldsymbol{b}_i), \tag{3.2}$$

where a_i are known as activation functions. Typically, hidden layers use nonlinear functions, such as Sigmoid, hyperbolic tangent or Rectified Linear Unit (ReLU) [18]. The ReLU activation $a(x) = \max(0, x)$ improves the training speed and solves the vanishing gradient problem. The output layer often uses linear and Sigmoid (Softmax) activations in regression and classification tasks, respectively. The matrices $\boldsymbol{\Theta}_i$ represents the weights of the i-th layer, which has a dimensionality of $W_i \times W_{i-1}$. The weights are the essence of the expressive power of neural networks. The training loop optimizes them during the learning process, as described in Section 3.1.3.

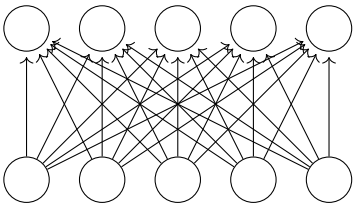
3.1.2. Convolutional neural networks

Convolutional neural networks (CNN) were among the earliest examples of successful deep NNs ever trained [20]. Few researchers independently introduced CNNs in close timeframes and with similar characteristics. Earlier, we had the Neocognitron [15], where multiple layers of local feature detectors share weights. Second, LeCun et al. [34] deployed a convolutional network to recognise handwritten characters. Additionally, Time-delay neural networks proposed in [58] presented a similar architecture to convolutional networks but applied to audio signals in a moving window. Even if CNNs have been present for nearly three decades, real breakthroughs happened when the community started to employ GPUs for large image datasets to solve large-scale object classification tasks. For example, among the early breakthroughs, the 8-layers AlexNet [32] was trained with millions of weights over 1 million pictures from the ImageNet dataset [52] on graphic cards. Since then, researcher have proposed many deeper convolutional networks [23, 51, 55].

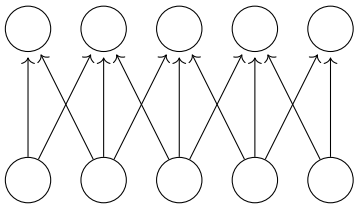
Nowadays, the deep learning community defines a CNN as a particular neural network architecture containing at least one convolutional layer [20]. Structurally, we can coincive a convolutional layer as an FC layer that shares weights across all the neurons, and its local connections reduce the number of parameters by not connecting all the inputs with the neuron[2]. Consequently, the number of parameters drastically decreases in convolutional layers. The Figure 3.2 illustrates the differences between a 1D convolutional layer and an FC one.

Mathematically, we can compute the output of a convolutional layer $\boldsymbol{o}$ given an input

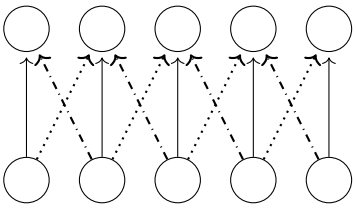
[2]This is true if the convolutional layer has only one kernel. Otherwise, each kernel represents a different aforementioned FC-like layer.



(a) FC layer: 25 weights.



(b) Layer with local connections: 13 weights.



(c) Convolutional layer, i.e. local shared connections: 3 weights.

Figure 3.2: Weights comparison among classical FC layer, layer with local connections, and convolutional layer.

$\boldsymbol{x}$ and the parameters $\boldsymbol{\theta}$, named kernel, as

$$\boldsymbol{o} = a\left[\begin{pmatrix} g_1(\boldsymbol{x};\,\boldsymbol{\theta}) & \dots & g_{W_i}(\boldsymbol{x};\,\boldsymbol{\theta}) \end{pmatrix}\right] + \boldsymbol{b}, \tag{3.3}$$

$$g_j(\boldsymbol{x};\,\boldsymbol{\theta}) = \sum_m x_{j+m} \cdot \theta_m, \tag{3.4}$$

where W_i is indeed the size of the kernel, $a(\cdot)$ is the activation function, typically ReLU, and $\boldsymbol{b}$ is the bias term. The function g_j recall the discrete convolutional operator $o[t] = (\theta \star x)[t] = \sum_m x[t-m]\theta[m]$, but it is actually implemented in most DL framework as a discrete cross-correlation operator.

We can further generalise the convolutional layer in multiple ways. Firstly, the input could contain additional dimensions, called input channels C_{in}. In this case, we sum all the kernel cross-correlation responses. This is particularly helpful for 2D convolutional neural networks to process colourful images because we can consider each RGB planes as an additional channel. Secondly, the layer outputs can contain additional dimensions too, namely output channels C_{out}. On the one hand, we need these in hidden layers to propagate different feature maps concurrently. On the other hand, we necessitate channels to encode additional information in the overall network outputs, such as in image segmentation tasks with fully-convolutional neural networks [37]. In this example, the locality information spans across each separated pixel of the image. Therefore, there will be supplementary

kernel weights for each input and output channel. For this reason, Formula 3.3 becomes

$$\boldsymbol{o}_k = a\left[\left(g_{1,k}(\boldsymbol{X};\,\boldsymbol{\Theta}) \quad \dots \quad g_{W_i,k}(\boldsymbol{X};\,\boldsymbol{\Theta})\right)\right] + \boldsymbol{b}_k, \tag{3.5}$$

$$g_{j,k}(\boldsymbol{X};\,\boldsymbol{\Theta}) = \sum_{c}^{C_{\text{in}}} \sum_{m} X_{j+m,c} \cdot \Theta_{m,c,k}, \tag{3.6}$$

where $k = 1, \dots, C_{\text{out}}$.

Typically, downsampling layers, known as poolings, interleaves convolutional layers in the overall architecture. These layers present small non-overlapping identity kernels that fit the selected input cells to aggregators, such as maximum or average for Max and Average pooling, respectively. Pooling layers serve not only to reduce computations, but also to shrink the input size that consequently helps to prevent overfitting [9]. Therefore, CNNs consists of stack of blocks with one or more convolution layers and possible poolings. Their eventual feature map undergoes flattering and flows to FC layers. These last layers exploit the representation learnt by the convolutional layers, and perform the actual classification or regression task.

3.1.3. Training a neural network: cost functions

Previous sections described the mathematical formulae that neural network layers evaluate. The calculations depend on the value of the weights that layers contain. A NN can *learn* the weight values by tweaking them in the training process, during which an algorithm optimizes them from the training set samples.

Regarding supervised learning tasks, the training process consists in minimizing a cost function $\mathcal{L}(\boldsymbol{\theta})$, also known as loss function, with respect to the network parameters θ given a training set $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}$. In regression tasks, such as the one addressed in this thesis, common loss functions includes Mean Absolute Error (MAE) and (Root) Mean Squared Error (R)MSE. Considering the MAE, the loss function is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} |\boldsymbol{y}^{(i)} - f(\boldsymbol{x}^{(i)};\,\boldsymbol{\theta})|, \tag{3.7}$$

where f is the function that the NN computes. Gradient-descent-based techniques can optimize $\mathcal{L}$. Basically, the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ of the loss function indicates the map of direction in the parameter space where the loss function is growing. Hence, given an initial point in the parameter space $\boldsymbol{\theta}_0$, the optimal value can be approached by iteratively subtracting

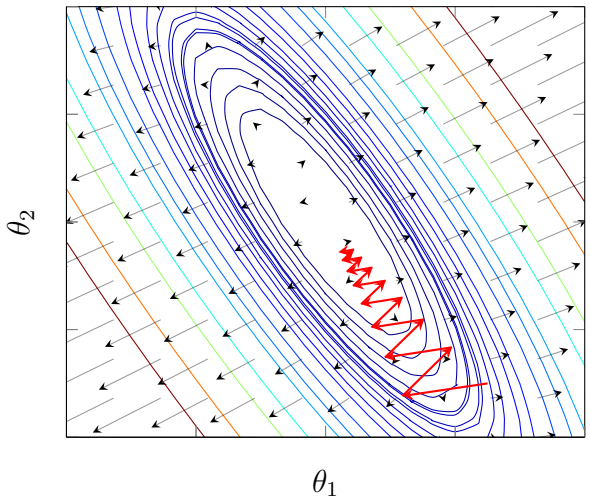


Figure 3.3: Illustration of the gradient descent iterations. A bivariate cost function defined by the level curves in the 2D parameter space. The vector field shows the gradient of the function in several points. The red arrows indicate the sequence of values reached during the iterations, until a minimum is approached.

positive factors that are opposite to the gradient direction:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t+1} - \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}|_{\boldsymbol{\theta}_t}, \tag{3.8}$$

where the factor η is known as *learning rate* and represent the step size of each iteration. Figure 3.3 illustrates the process in a simplified instance.

To train NN via gradient descent, we can adopt the backpropagation algorithm to compute the gradients of each parameter of the NN layers. Historically, many researchers proposed and rediscovered the backpropagation [36, 60]. However, the modern version of this technique spread in the deep learning community only after [46]. As the name suggest, this technique consists in propagating the errors between the outputs of the network and the target values throughout all the connections of the hidden layers. If we consider neural networks defined as function compositions, like in (3.1), the backpropagation resorts to applying the chain rule of derivatives to each layer function f_i, i.e.

$$\boldsymbol{y}^{(i)} = (f_1 \circ \ldots \circ f_i)(\boldsymbol{x}), \quad i = 1, \ldots, D, \quad \boldsymbol{y}^{(0)} = \boldsymbol{x} \tag{3.9}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_j^{(i)}} = \sum_{k=1}^{W_i} \frac{\partial \mathcal{L}}{\partial y_k^{(i)}} \frac{\partial y_k^{(i)}}{\partial \theta_j^{(i)}} \tag{3.10}$$

$$\frac{\partial \mathcal{L}}{\partial y_j^{(i-1)}} = \sum_{k=1}^{W_i} \frac{\partial \mathcal{L}}{\partial y_k^{(i)}} \frac{\partial y_k^{(i)}}{\partial y_j^{(i-1)}}, \tag{3.11}$$

where $\theta_j^{(i)}$ is a generic weight or bias parameter of the layer i, $y_k^{(i)}$ the outputs of the layer i, and $y_k^{(i-1)}$ the inputs of the layer i. The backpropagation is a powerful technique that can potentially support any kind of layer. It is enough for a new layer i to define the computations for forward function f_i from inputs and weights, as well as the backward computations that propagate the derivative with respect to the outputs $\partial\mathcal{L}/\partial y_k^{(i+1)}$ into the derivatives of inputs $\partial\mathcal{L}/\partial y_k^{(i-1)}$ and weights $\partial\mathcal{L}/\partial\theta_k^{(i)}$, i.e. equivalent to define $\partial y_k^{(i-1)}/\partial y_k^{(i)}$ and $\partial y_k^{(i+1)}/\partial\theta_k^{(i)}$ respectively.

Improved gradient descent

The gradient descent optimization as presented in Equation (3.8) is not popular nowadays, due to its long convergence time. As a matter of fact, various researches have been focusing on improving the gradient descent following different methods.

Firstly, the loss function is actually calculated on randomized mini-batches $\mathbb{B}_i$, i.e. disjoint portions of the training set, whose union is equal to the training set. A gradient is approximated from one batch, then used to update the weights:

$$\frac{\partial\mathcal{L}}{\partial\theta} \approx \frac{1}{|\mathbb{B}_i|}\sum_{n\in\mathbb{B}_i}\frac{\partial}{\partial\theta}g_\theta\left(\boldsymbol{y}^{(n)},\widehat{\boldsymbol{y}}^{(n)}\right). \tag{3.12}$$

This technique, named stochastic gradient descent (SGD), indirectly injects noise in the gradient calculation. SGD helps mitigate redundancy that the training set might contain, by avoiding updating the weights with similar values of the gradient. The NN sees every mini-batch once, in sequence. When the NN saw all the training samples through mini-batches, an *epoch* concluded, and the cycle can start again. The batch size needs to be balanced with the computation efficiency, as smaller batch sizes are less computationally efficient.

As inconvenience of mini-batch training, it might happen that the injected noise could resort to non-zero gradient estimation even in proximity of the optimum. Two techniques alleviate this problem. The learning rate could be reduced every τ iterations, namely *learning rate annealing*. Moreover, weights could be updated by exponential moving average $\theta_t^{'} = \gamma\theta_{t-1}^{'} + (1-\gamma)\theta_t$.

Moving averages are also employed in improved optimizers, along with the Momentum technique. Referring to the weight optimization as a ball following the red arrows in Figure 3.3, the Momentum [42] helps reduce oscillations in directions of high curvature of $\mathcal{L}_\theta$, and prefer directions with a more consistent gradient [57]. The Momentum achieves it

by combining negative gradients in the speed v_t and updating the weights to

$$v_{t+1} = \alpha v_t - \eta_t \frac{\partial \mathcal{L}}{\partial \theta_t} \tag{3.13}$$

$$\theta_{t+1} = \theta_t + v_{t+1}, \tag{3.14}$$

where $\alpha < 1$ is the momentum coefficient, a hyperparameter responsible for the velocity decay. If the Momentum is close to 1, then the descent is faster than the simple one. Hence, it makes sense to start with values like $\alpha \approx 0.5$, and in later epochs adapt it to higher values, like 0.9 or 0.99, especially the training loss reaches a plateau. One variation to this technique is the *Nesterov Momentum*, defined in [54] and inspired by [40]. This technique updates the gradient first, then refreshes the speed with the new gradient value. Intuitively, the Nesterov Momentum tries to correct the direction mistakes after they have happened, while the plain Momentum attempts to do it beforehand [20].

Especially with very deep NN, the magnitude of the gradients often differs across different layers. Hence, modern optimizers employ independent learning rates for each weight. They achieve this by multiplying a global learning rate by local coefficients computed empirically for each weight. This is the case for RMSProp [57], an improved version of RProp [45] for multi-batch training. RMSProp divides the gradient by a running average of the latest gradient intensities, i.e.

$$\theta_t = \theta_{t-1} - \eta_t \frac{g_t}{\sqrt{v_t} + \epsilon}, \tag{3.15}$$

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2. \tag{3.16}$$

Similarly, Adam [29] combines RMSProp and AdaGrad [10]. Adam extends them by substituting the gradient with the exponential moving average of the gradient, as well as by correcting the estimation of the first and second-order moments with

$$\theta_t = \theta_{t-1} - \eta_t \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}, \tag{3.17}$$

$$\begin{aligned} \widehat{m}_t &= {}^{m_t}/_{1-\beta_1^t}, \quad m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ \widehat{v}_t &= {}^{v_t}/_{1-\beta_2^t}, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \end{aligned} \tag{3.18}$$

In practice, Adam often works well [20], because it defines a bound for the step size $\Delta\theta_t$, thus steps are never too large. Additionally, Adam automatically anneals the learning rate as approaches convergence, so the optimized parameters fluctuate around the optimal values.

3.2. Relevant studies

This section reviews general deep learning techniques employed in the solution of inverse problems, as well as previous studies in the inversion of the SC generation.

3.2.1. Deep learning in inverse problems

As with many other challenges nowadays, solutions to inverse problems also find benefits in recent advancements of DL [41]. Generally, solving an inverse problem consists of determining the problem causes x when we can only measure effects y [31]. The effects y are typically observed from the output of a forward process $\mathcal{F}(x)$. Normally $\mathcal{F}$ is nonlinear and not invertible. This makes inverse problems impossible to solve without resorting to data-driven methods, such as deep learning. The data-driven methods can address an inverse problem by learning the backward process $\mathcal{B} = \mathcal{F}^{-1}$ or by optimizing the solution $\widehat{x}_0$ of a particular instance y_0.

Supervised deep learning methods solve inverse problems by exploiting paired samples $\{(x, y)\}$ available from the forward process $y = \mathcal{F}(x)$. The causes become neural networks targets to be predicted, while the effects become inputs. Then the network is trained by optimizing a chosen loss function over the targets x. On the other hand, if $\mathcal{F}$ is computationally feasible, the training procedure could embed it, and penalise the error in the forward domain, i.e. the error between the effects y and $\mathcal{F}(x)$.

For example, researchers successfully adopt deep learning as explained above to solve imaging inverse problems, commonly present in various scientific disciplines, such as Medical Imaging [30] and Geophysical Imaging [3]. In these inverse problems, x and y often have the same dimensionality, e.g., in tasks like denoising and deblurring, or similar dimensionality in tasks like superresolution or undersampled reconstructions.

This thesis focus on a particular example of parameter estimation inverse problems [56], i.e. the SC generation. Generally, in parameter estimation inverse problems, the causes space has much lower dimensionality than the effects one, i.e. few parameters of the forward function lead to vectors of higher dimension, such as one-dimensional signals. In our problem, as discussed in Chapter 2, the $\mathcal{F}$ is only available at test time, hence we cannot embed it at training time. Similarly, other inverse problems have employed NNs to solve parameter estimations. For instance, researchers trained a fully-connected neural network to predict the parameters that describe the shape of violins, given their desired frequencies [48]. In Fluid Dynamics, researchers in [50] show how to predict via NN the airfoil shape parameters given the velocity maps of air flowing throughout. In Astronomy, the gravitation waves detected during specific events, such as binary black

holes merging, have been used to predict characteristics of the involved stars via Variational Autoencoders [16] and Autoregressive NN flows [21]. Differently from the method proposed in this thesis, these works enforced no properties on the effect domain at training time.

Similarly to the weighted scheme adopted in our method, other works thought about enforcing properties over the forward domain. These works include Physics Informed NN (PINN) [22, 44] and Theory guided NN (TgNN) [59]. Both techniques require that the differential equation describing the forward model be available. They act as a regularization method to inverse problem ill-conditions, by embedding the computation of the differential equations in the NN training loops. Differently, our method requires no known expression of the differential equation. In principle, the dataset of pairs would be enough.

3.2.2. Data-driven methods for supercontinuum generation

Machine learning and search optimization techniques are accelerating the research and development of ultrafast photonics [17]. In particular, several works have attempted to optimize structures and components of the SC spectrum generation, shown in Figure 1.1 in Chapter 2.

Some researchers have extensively adopted Genetic algorithms (GAs) [25] as an optimization technique for ultrafast photonics. Historically, GAs have been employed to design the structure of the optic fibre [28, 39, 62, 63]. For example, the optic fibres considered in those works consist of crystal fibres, with concentric rings of air-holes. Therefore, the method mainly optimizes the rings, air-holes radius, and air-hole distance by minimizing appropriate fitness functions that model desired properties, such as chromatic dispersion, power gain, fibre robustness to production fluctuations.

In our work, we consider the design of the optic fibre as fixed, so the dispersion model is common to every generated spectrum. On the other hand, we focus on the ultrafast train pulses of light that enter the fibre. Variations of pulse parameters still lead to nonlinear variations in the eventual spectrum obtained at the fibre end. Similarly to our work, previous researches focused on optimizing the pulse parameters to enforce specific properties in the output spectrum. For instance, Raman frequency conversion maximization was demonstrated in [4]. This research adopted a GA to find the parameters we also considered in this thesis that maximize the spectrum energy in a given wavelength range. Previously, other authors proposed a solution with a similar method [38]. These two works especially focus on optical coherence tomography (OCT), where most of the near and mid-infrared (IR) regions requires most of the power.

However, GA are computationally expensive in time. The work in [38] showed that grid computing reduces GA convergence time from 10 hours of a single computer, to 90

minutes. In [4], they show that GA still reach better accuracies than exhaustive searches. However, GA-based techniques do not output a model for point estimation. Therefore, if users provide new spectrum characteristics, a new GA search is required.

Contrarily, DL techniques optimize models that can relatively quickly predict parameters for new target characteristics. In our work, we train deep learning models that given a spectrum, predict the best parameters that generate a spectrum as close as possible to the input one. This is an inverse problem, where the forward process is the SC generation. The forward process is computationally expensive and nonlinear, therefore cannot be inverted analytically. On one hand, the work in [47] shows how recurrent neural networks (RNN) with long short-term memory (LSTM) [24] can predict such complex dynamics. On the other hand, to the best of our knowledge, this research is the first work where DL techniques are employed to solve the inverse SC generation.

3.3. Function approximation

We define a generic training set as a set of pair of vectors $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}$, where $\boldsymbol{x} \in \mathbb{R}^D$ is the input value, $\boldsymbol{y} \in \mathbb{R}^C$ is the target value, D and $C \in \mathbb{N}$. The target value is associated to the input by the relation $\boldsymbol{y}^{(i)} = f(\boldsymbol{x}^{(i)}) + \boldsymbol{\epsilon}^{(i)}$, where $f : \mathbb{R}^D \to \mathbb{R}^C$ and $\boldsymbol{\epsilon}^{(i)}$ is the unknown noise for the sample i-th.

The goal of function approximation tasks is to compute a suitable approximation of f. The term *suitable* depends on the required quality of the approximation. We can assess the quality on another set, namely the testing set, by measuring the displacements of approximated values and testing targets, accordingly to a designated distance function. Normally, solving the task via direct usage of function f is disallowed, since f might be unknown or extremely expensive to compute in terms of resources. The approximation can only leverage the training set.

Several techniques solve the function approximation task. For instance, we could see it as a supervised learning task, that machine learning and deep learning techniques can address. However, this section introduces two algebraic linear techniques, i.e. polynomial approximation (PA) and local polynomial approximation (LPA). We apply these techniques extensively during this thesis work to approximate the spectrum loss function via weighted loss.

In the next sections, without loss of generality, we simplify the notation and consider 1-dimensional inputs and targets.

3.3.1. Algebraic definitions

This section provides definitions around the concept of the *dual frame*, i.e. the matrix required to perform the polynomial approximation in Section 3.3.2 and beyond.

Definition 3.1 (A Dual Frame). *Given a set of vectors* $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$*, namely the frame, and* $\boldsymbol{v}, \boldsymbol{w}$ *some vectors in the same space, there exists another set of vectors* $\{\widetilde{\boldsymbol{v}}_1, \dots, \widetilde{\boldsymbol{v}}_n\}$*, namely a dual frame, such that* $\operatorname{span}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\} = \operatorname{span}\{\widetilde{\boldsymbol{v}}_1, \dots, \widetilde{\boldsymbol{v}}_n\}$ *and*

$$\sum_{i=1}^{n} \langle \boldsymbol{v}, \boldsymbol{v}_i \rangle \widetilde{v}_i = \sum_{i=1}^{n} \langle \boldsymbol{v}, \widetilde{\boldsymbol{v}}_i \rangle \boldsymbol{v}_i = \underset{\boldsymbol{w} \in \operatorname{span}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}}{\arg\min} \|\boldsymbol{v} - \boldsymbol{w}\|. \tag{3.19}$$

Formally, there might many dual frame for a specified set of vectors that satisfy the Definition 3.1. Therefore, we provide a way to select the one required, as well as a convenient formula to calculate it.

Definition 3.2 (The Dual Frame). *When there exists multiple dual frame, i.e.* $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ *are not linearly independent, the dual frame is defined as the one that minimizes* $\sum_{i=1}^{n} |\langle \boldsymbol{v}, \widetilde{\boldsymbol{v}}_i \rangle|^2$.

Definition 3.3 (The Dual Frame Matrix). *The dual frame matrix* $\widetilde{\boldsymbol{V}}$ *contains the dual frame vectors* $\{\widetilde{\boldsymbol{v}}_i\}$ *as column. Given the frame matrix* $\boldsymbol{V}$ *containing the frame vectors as columns, the dual frame matrix can be computed as:*

$$\widetilde{\boldsymbol{V}} = \boldsymbol{V}(\boldsymbol{V}^\top \boldsymbol{V})^\dagger, \tag{3.20}$$

where $\dagger$ *denotes the pseudo-inversion operator.*

3.3.2. Polynomial approximation

Polynomial approximation (PA) tries to approximate functions by computing the best coefficients θ_j, $j = 0 \dots M$, where M is the polynomial degree. We can express the approximation with

$$\hat{y}(x; \boldsymbol{\theta}) = \sum_{j=0}^{M} \theta_j \cdot b_j(x), \quad b_j(x) = \frac{x^j}{j!}. \tag{3.21}$$

This model is non-linear with respect to the input variable x, however it is linear with respect to the coefficients θ_i.

We now analyse the polynomial approximation from an algebraic perspective. Given the column vectors $\boldsymbol{x} = \begin{pmatrix} x_1 & x_2 & \dots & x_N \end{pmatrix}^\top$ and $\boldsymbol{y} = \begin{pmatrix} y_1 & y_2 & \dots & y_N \end{pmatrix}^\top$, we consider the well-known N-dimensional vector space of M-degree polynomials individuated by the set of vectors $\{b_j(\boldsymbol{x})\}_{j=0}^{M} \subset (\mathbb{R}^N, \langle \cdot, \cdot \rangle)$, where $(\mathbb{R}^N, \langle \cdot, \cdot \rangle)$ is an N-dimensional real vector

space with standard inner product. We can now rewrite the polynomial approximation $\widehat{\boldsymbol{y}}$ as element of the polynomial vector space, i.e. vector $\boldsymbol{x}$ and rewrite Equation (3.21) as:

$$\boldsymbol{A} = \begin{bmatrix} b_0(\boldsymbol{x}) & b_1(\boldsymbol{x}) & \dots & b_M(\boldsymbol{x}) \end{bmatrix} \tag{3.22}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{A} \cdot \boldsymbol{\theta}. \tag{3.23}$$

Notice that, the matrix $\boldsymbol{A}$ collect the set of vectors $\{b_j(\boldsymbol{x})\}$, where $b_j(\cdot)$ is applied element-wisely. The vector set $\{b_j(\boldsymbol{x})\}$ is then called frame, and $\boldsymbol{A}$ is called frame matrix [12]. Consequently, we can see $\boldsymbol{A}$ as synthesis operator, because multiplied by the coefficient vector $\boldsymbol{\theta}$, $\boldsymbol{A\theta}$ produces a linear combination of the vector set $\{b_j(\boldsymbol{x})\}$.

The PA problem reduces in finding the best coefficient vector $\boldsymbol{\theta}$ such that minimizes the distance of $\hat{\boldsymbol{y}}$ from $\boldsymbol{y}$:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|. \tag{3.24}$$

From an algebraic perspective, this is equivalent to employing the dual frame $\widetilde{\boldsymbol{A}}$ of the frame $\boldsymbol{A}$ from Equation (3.20) as an analysis operator, i.e.

$$\boldsymbol{\theta} = \widetilde{\boldsymbol{A}}^{\top}\boldsymbol{y}, \quad \widetilde{\boldsymbol{A}} = \begin{bmatrix} \widetilde{\boldsymbol{b}}_0 & \widetilde{\boldsymbol{b}}_1 & \dots & \widetilde{\boldsymbol{b}}_M \end{bmatrix} \tag{3.25}$$

To perform the approximation on a sample x_0, we sufficiently synthesize the coefficients $\boldsymbol{\theta}$ via frame matrix, i.e.

$$\hat{y}(x_0) = \sum_{j=0}^{M} \langle \boldsymbol{y}, \widetilde{\boldsymbol{b}}_j \rangle b_j(x_0) \tag{3.26}$$

$$= \langle \boldsymbol{y}, \sum_{j=0}^{M} \widetilde{\boldsymbol{b}}_j b_j(x_0) \rangle \tag{3.27}$$

$$= \langle \boldsymbol{y}, \widetilde{\boldsymbol{A}}\overline{\boldsymbol{b}}^{(x_0)} \rangle \tag{3.28}$$

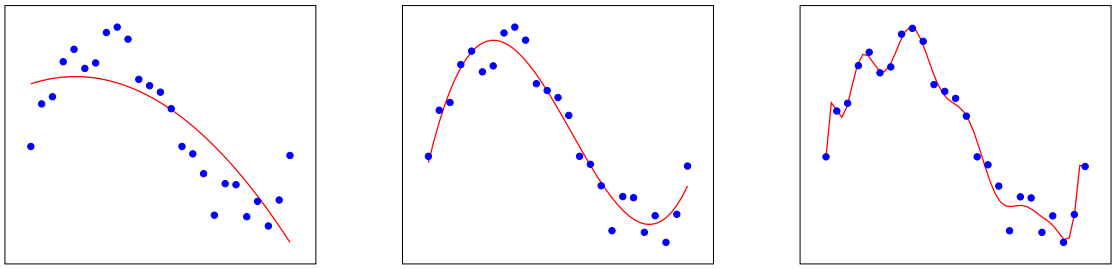
$$= \langle \boldsymbol{y}, \boldsymbol{g}(x_0) \rangle, \tag{3.29}$$

where $\overline{\boldsymbol{b}}^{(x_0)} = \begin{pmatrix} b_0(x_0) & b_1(x_0) & \dots & b_M(x_0) \end{pmatrix}^{\top}$, $\boldsymbol{g}(x_0) = \widetilde{\boldsymbol{A}}\overline{\boldsymbol{b}}^{(x_0)}$ is the kernel function.

The choice of the hyperparameter M, i.e. the degree of the polynomial, represents a trade-off between bias and variance, as it can be seen in Figure 3.4.

3.3.3. Local polynomial approximation

Local polynomial approximation (LPA) creates a function approximation, whose expression differs in each point to estimate. LPA achieves this by selecting a different subset of the training set in each point. In this way, we can obtain a more precise function approximation



(a) Polynomial degree 2, underfitting. (b) Polynomial degree 3. (c) Polynomial degree 16, overfitting.

Figure 3.4: Different choices of the polynomial degree

in the neighbourhood of the local point.

Given the point x_0 where to estimate the function, one possible method consists of selecting the P pairs whose input variable is closer to x_0. We can further generalize this approach by taking a smooth Gaussian window centred in x_0, instead of a crisp window. Therefore, we can consider the contribution of more points, but only the closer ones have higher impact. We express this by assigning a localization coefficient to each pair (x_i, y_i), e.g.

$$c_i = e^{-\alpha \|x_i - x_0\|_2^2}, \tag{3.30}$$

where α controls the dispersion of the Gaussian window. Therefore, we can update the definition of the kernel function $\boldsymbol{g}(x_0)$ from Equation (3.29) to include the localization coefficients as

$$\boldsymbol{g}_{\boldsymbol{C}}(x_0) = \boldsymbol{C}\widetilde{\boldsymbol{A}}_W \overline{\boldsymbol{b}}(x_0), \quad \boldsymbol{C} = \mathrm{diag}(c_1, \ldots, c_N), \tag{3.31}$$

and in contrast with the dual frame with Euclidean distance of Formula 3.20, $\widetilde{\boldsymbol{A}}_C$ denotes the dual frame with the induced weighted l_2 norm:

$$\widetilde{\boldsymbol{A}}_W = \boldsymbol{A}(\boldsymbol{A}^\top \boldsymbol{C} \boldsymbol{A})^\dagger. \tag{3.32}$$

Further generalization can be achieved with adaptive LPA methods, i.e. windows can broaden or shrink automatically depending on x_0 neighbourhood. For instance, we could resort to LPA-ICI [13], namely local polynomial approximation and intersection of confidence intervals. This anisotropic adaptive method automatically fits separated windows with respect to different direction ranges of the hyper-space. This approximation algorithm have shown state-of-the-art performances in other works [14, 61].

4 | Solving the SC inverse problem

This chapter provides details about our proposed solution. The first section gives an overview about the method, the data preprocessing and overall pipeline. Then the chapter continues giving details about the weighted loss function in Section 4.2 and the employed models for parameter estimation in Section 4.3.

4.1. Proposed solution overview

Referring to the problem formulation in Section 2.2, we train two kinds of DL models, namely FC in Section 4.3.1 and CNN in Section 4.3.2. As shown in Figure 4.1, the models receive as input the preprocessed generated spectra and as targets the pulse parameters that generated the input spectra. Hence, NNs could minimize a loss function $\mathcal{L}_{\mathbb{Y}}$ (4.7) over parameters. However, this is not enough representative of the problem. We would like to actually predict parameters that generate a spectrum as close as possible to the input one. In DL, this would translate to optimize a loss function $\mathcal{L}_{\mathbb{S}}$ (4.6) in the spectrum domain.

On the one hand, we hypothesize that limiting the learning process at the parameter domain would neglect an important constraint of the problem. On the other hand, as discussed in Section 5.2, it would be extremely expensive to generate the spectra from predicted parameters during every epoch of the backpropagation. To overcome this obstacle, we introduce a surrogate loss function $\mathcal{L}_W$ (4.8) that approximate the ideal loss $\mathcal{L}_{\mathbb{S}}$. This surrogate loss function can be adopted in the training with no additional computational costs during the backpropagation epochs. Nevertheless, once the training process completes, we still evaluate the loss $\mathcal{L}_{\mathbb{S}}$ as assessment metric. This require generating the spectra $\widehat{\boldsymbol{s}}$ from the model predictions $\widehat{\boldsymbol{y}}$ by following again the forward process. Figure 4.2 schematises further the data pipeline of the proposed solution and its evaluation process.

4.1.1. Binning spectra into areas

As shown in Figure 2.1, the generated spectra have a wide range of nonlinear oscillations that makes comparisons hard. Moreover, the dimensionality of $\mathbb{R}^N$, $N = 4096$, would be too computational intensive for the NN trainings. Therefore, domain experts suggests the

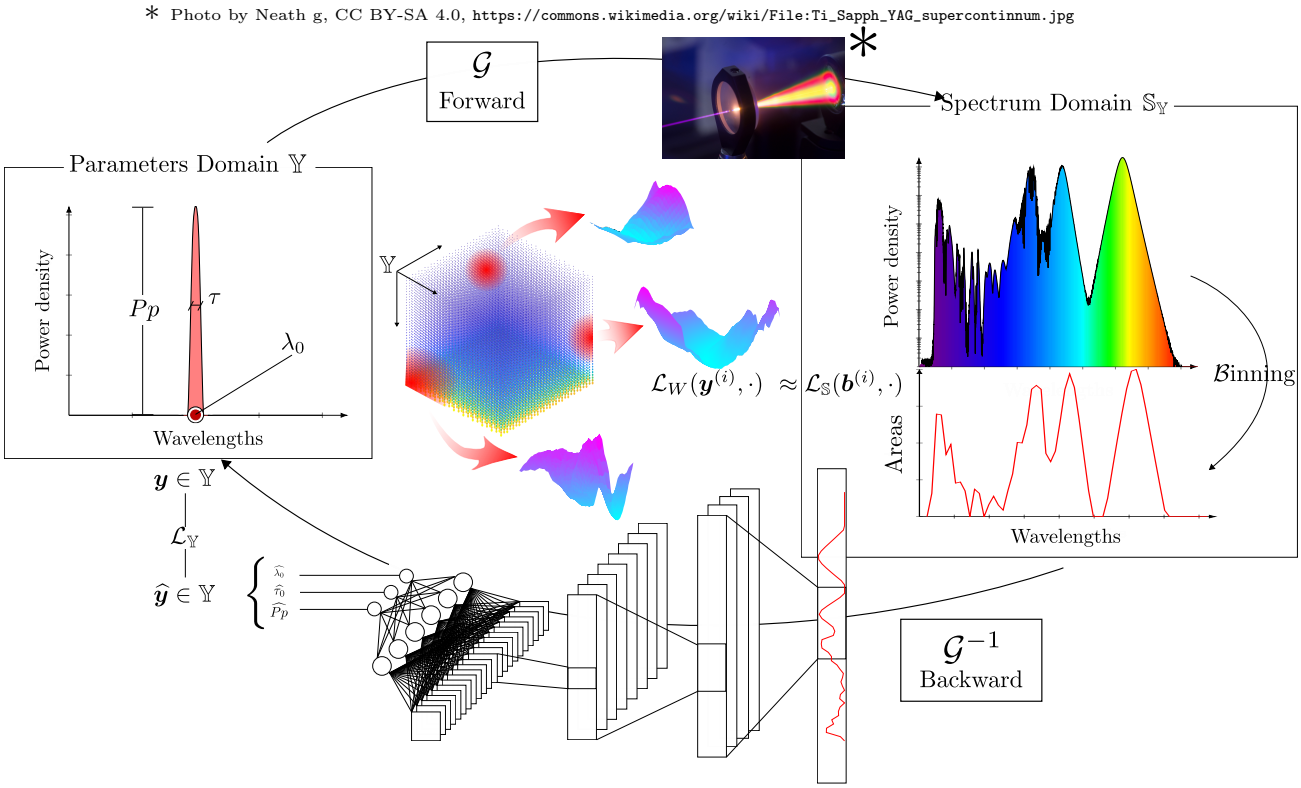


Figure 4.1: Overview of the problem and solution. SC generation represents the inverse problem forward pass from parameters to spectra. Fully-connected and convolutional networks learn the inverse process. From the parameters-spectrum pairs, we compute a weighted loss function $\mathcal{L}_W$ as nonparametric local polynomial approximation (LPA) of the spectrum loss $\mathcal{L}_{\mathbb{S}}$. We then emply the weighted loss function $\mathcal{L}_W$ to train the networks.

integral binning as possible method to downsample spectra. Given the number of bins n and the integration interval $I = (\lambda_a, \lambda_b) = \bigcup_{i=1}^{n} I_i$, the integral binning is an operator $\mathcal{B}_n^I : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^n$ defined as

$$\boldsymbol{a} = \mathcal{B}_n^I(\boldsymbol{\lambda}, \boldsymbol{\sigma}) = \left[\int_{I_i} \overline{\boldsymbol{s}}(\omega) \, \mathrm{d}\omega \right]_{i=1}^{N}, \tag{4.1}$$

where $\overline{\boldsymbol{s}}(\omega) = (\overline{\boldsymbol{\lambda}}, \overline{\boldsymbol{\sigma}}) \in \mathbb{R}^{N+n+1} \times \mathbb{R}^{N+n+1}$ is the enhanced representation of the spectrum $\boldsymbol{s} = (\boldsymbol{\lambda}, \boldsymbol{\sigma})$, i.e. the spectrum is combined with the linear interpolation $\boldsymbol{\sigma}_I \in \mathbb{R}^{n+1}$ computed from $\boldsymbol{s}$ on the area grid $\boldsymbol{\lambda}_I \in \mathbb{R}^{n+1}$. Then we define the areas grid as

$$\boldsymbol{\lambda}_I = \left[\lambda_a + (i-1) \cdot \tfrac{\lambda_b - \lambda_a}{n} \right]_{i=1}^{n+1}. \tag{4.2}$$

We practically compute the integral of Equation 4.1 via trapezoidal numerical integration Next, we consider the cumulative sum, resorting to the cumulative areas vector $\overline{\boldsymbol{\alpha}} \in \mathbb{R}^{N+n+1}$

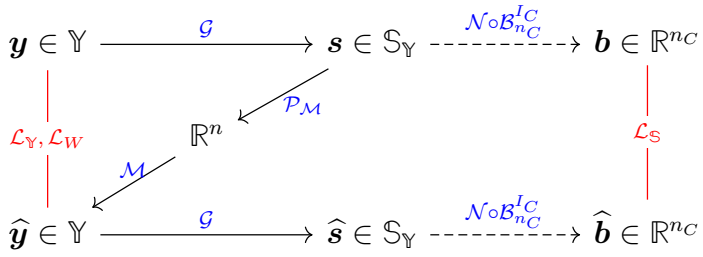


Figure 4.2: Data pipeline. In this schema, a model $\mathcal{M}$, e.g. FC or CNN, is trained optimizing $\mathcal{L}_W$ or $\mathcal{L}_Y$ given spectra-parameters pairs $\{(\boldsymbol{y}, \boldsymbol{s})\}$. The spectra undergo the preprocessing $\mathcal{P}_{\mathcal{M}}$, that depends on the model $\mathcal{M}$. Binned areas computed from training spectra $\boldsymbol{b}$ and regenerated spectra $\widehat{\boldsymbol{b}}$ from model predictions $\widehat{\boldsymbol{y}}$ are finally compared through $\mathcal{L}_{\mathbb{S}}$.

such that

$$\overline{\alpha}_j = \frac{\overline{\lambda}_j - \overline{\lambda}_{j-1}}{2} \left(\overline{\sigma}_j + \overline{\alpha}_{j-1}\right), \quad j = 2, \ldots, N + n + 1 \text{ and } \overline{\alpha}_1 = 0. \tag{4.3}$$

Finally, given $\boldsymbol{\alpha}_I$ the values of $\overline{\boldsymbol{\alpha}}$ corresponding to $\boldsymbol{\lambda}_I$, the area vector $\boldsymbol{a}$ is defined as the difference of adjacent cumulative areas $\boldsymbol{\alpha}_I$ on the area grid $\boldsymbol{\lambda}_I$. We can easily calculate it with a convolution such as

$$\boldsymbol{a} = \boldsymbol{\alpha}_I \star \begin{bmatrix} 1 & -1 \end{bmatrix}. \tag{4.4}$$

The Binning operator is quite flexible, since different options for the integration intervals and number of bins can be provided. For instance, the CNN model can afford a larger number of bins than the FC, as their tolerance to the input vector dimensionality is different. Moreover, the operator is helpful because provides a common grid where we can compare spectra integrated in the same intervals I. As a matter of fact, spectra generated by the simulator always have different wavelength grid given different parameters. In our assessment, we adopted a common interval of $I_C = (1043.7, 2675.4)$ nm and a common number of bins $n_C = 60$ to compare ground-truth spectra with the spectra generated by the estimated parameters.

4.1.2. Normalization and Decibel transformation

Binned areas have much larger magnitude of normal spectra. In addition to that, the scale of different bins varies considerably both in the same spectrum or among different ones. From a physical point of view, it is not necessarily important to match the scale in generate spectra as it is for the energy distribution. Therefore, experts recommend

converting the areas to dB scale and normalize in the range of -40 dB and 0:

$$\boldsymbol{b} = \mathcal{N}(\boldsymbol{a}) = \max\{-40,\ 10\log_{10}(\boldsymbol{a}) - 10\log_{10}(\max(\boldsymbol{a}))\}. \tag{4.5}$$

The clipping of -40 dB is chosen to ignore small-scale values, considered as noise.

4.2. Loss functions

As explained in Section 3.1.3, NNs necessitate to optimize loss function. This section describes the ideal loss $\mathcal{L}_{\mathbb{S}}$ in the spectra domain, the problem of a loss in the parameters domain $\mathcal{L}_{\mathbb{Y}}$, and the rationale behind the surrogate loss $\mathcal{L}_W$.

In the following subsections, all the loss function expressions consider single input-target samples as argument, to simplify the notation. However, the training of the NNs actually computes the gradients from the sum of the losses applied in any sample of the batch $\mathbb{B}$, i.e. as explained in Equation (3.12).

4.2.1. Ideal loss

Ideally, we would like to compute the error between spectra, and propagate it back to the networks to update their weights. Hence, the ideal loss on the spectra domain is expressed by

$$\mathcal{L}_{\mathbb{S}}(\boldsymbol{b}, \widehat{\boldsymbol{b}}) = \frac{1}{R}\sum_{j=1}^{R}|b_j - \widehat{b}_j|, \tag{4.6}$$

where $\boldsymbol{b} = \mathcal{B}_{n_C}^{I_C}(\mathcal{G}(\boldsymbol{y}))$, $\widehat{\boldsymbol{b}} = \mathcal{B}_{n_C}^{I_C}(\mathcal{G}(\mathcal{M}(\boldsymbol{s})))$ involves the invocation of the generator, thus infeasible at training time. To circumvent the problem, we could propagate the error over the parameters $\mathcal{L}_{\mathbb{Y}}$, namely

$$\mathcal{L}_{\mathbb{Y}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{P}\sum_{j=1}^{P}|y_j - \widehat{y}_j|, \tag{4.7}$$

where $\widehat{\boldsymbol{y}} = \mathcal{M}(\boldsymbol{s})$ are the parameter predicted by the NN. This loss function has minimum in $\widehat{\boldsymbol{y}} = \boldsymbol{y}$, that is the same minimum of $\mathcal{L}_{\mathbb{S}}$. However, the neighbourhood of the minimum of $\mathcal{L}_{\mathbb{S}}$ diverges from the one in $\mathcal{L}_{\mathbb{Y}}$, as we can notice from Figure 4.3. Moreover, while the spectrum errors $\delta_j^{\mathbb{S}} = |b_j - \widehat{b}_j|$ would correct the networks giving feedback when the target spectrum and the generated from prediction has displacements, the parameter errors $\delta_j^{\mathbb{Y}} = |y_j - \widehat{y}_j|$ offer no information about that. The parameter loss is isotropic with respect to spectra shape in Formula 4.7.

4.2.2. Weighted loss as local polynomial approximation

To address the limitation of $\mathcal{L}_{\mathbb{Y}}$ explained in the previous section, we define a weighted loss function $\mathcal{L}_W$. The loss $\mathcal{L}_W$ consists of a nonparametric multivariate LPA of $\mathcal{L}_{\mathbb{S}}$. Being nonparametric means that for each sample $\left(\boldsymbol{b}^{(i)}, \boldsymbol{y}^{(i)}\right)$ we would have a different loss function expression $\mathcal{L}_W(\boldsymbol{y}^{(i)}, \cdot)$ such as

$$\mathcal{L}_W(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}) = \frac{1}{2}\sqrt{\boldsymbol{w}^{(i)} \cdot (\boldsymbol{y}^{(i)} - \widehat{\boldsymbol{y}}^{(i)})^2}. \tag{4.8}$$

Therefore, we compute the weighted loss by calculating multiple function approximations, one for each sample of the trainset. We calculate the function approximation as explained in Chapter 3. Specifically, for each reference sample $\left(\boldsymbol{b}^{(r)}, \boldsymbol{y}^{(r)}\right)$, we solve a LPA of $\mathcal{L}_{\mathbb{S}}(\boldsymbol{b}^r, \cdot)$ with $\mathcal{L}_W(\boldsymbol{y}^{(r)}, \cdot)$. Being $\boldsymbol{\delta}^{(i)} = |\boldsymbol{y}^{(i)} - \boldsymbol{y}^{(r)}|$, vector function for the polynomial coefficients $\boldsymbol{p}(\boldsymbol{\delta}) = \left(\frac{1}{2}\delta_1^2 \quad \delta_1\delta_2 \quad \delta_1\delta_3 \quad \frac{1}{2}\delta_2^2 \quad \delta_2\delta_3 \quad \frac{1}{2}\delta_3^2\right)^\top$, we define the frame matrix as:

$$\boldsymbol{A} = \begin{bmatrix} p_1(\boldsymbol{\Delta}^{(1)}) & \dots & p_6(\boldsymbol{\Delta}^{(n)}) \end{bmatrix} = \begin{bmatrix} p_1(\boldsymbol{\delta}^{(1)}) & \dots & p_6(\boldsymbol{\delta}^{(1)}) \\ \vdots & \ddots & \vdots \\ p_1(\boldsymbol{\delta}^{(n)}) & \dots & p_6(\boldsymbol{\delta}^{(n)}) \end{bmatrix}, \tag{4.9}$$

then we have $\boldsymbol{l} = \left(\mathcal{L}_{\mathbb{S}}\left(\boldsymbol{b}^{(r)}, \boldsymbol{b}^{(1)}\right) \quad \dots \quad \mathcal{L}_{\mathbb{S}}\left(\boldsymbol{b}^{(r)}, \boldsymbol{b}^{(n)}\right)\right)^\top$ and $\widehat{\boldsymbol{l}} = \boldsymbol{A}\boldsymbol{w}^{(r)}$. We want to find the vector of parameter weights $\boldsymbol{w}^{(r)}$ for the reference sample $\left(\boldsymbol{b}^{(r)}, \boldsymbol{y}^{(r)}\right)$ such that the distance between $\boldsymbol{l}$ and $\widehat{\boldsymbol{l}}$ is minimized. We want to do this with respect to the locality given by the Gaussian window:

$$\boldsymbol{w}^{(r)} = \arg\min_{\boldsymbol{w}} \{\boldsymbol{c}^{(r)} \cdot \left(\widehat{\boldsymbol{l}}^2 - \boldsymbol{l}^2\right)^2\}, \tag{4.10}$$

where $\boldsymbol{c}^{(r)}$ is a K-dimensional vector containing the localization coefficients, modelled by a Gaussian window with isotropic spread α. The coefficient c_i is expressed by $c_i^{(r)} = \exp\{-\alpha\|\boldsymbol{y}^{(r)} - \boldsymbol{y}^{(i)}\|^2\}$. The spread α has been chosen by cross-validation heuristics, and in our case it has been set to $\alpha = 1/6$.

Formula 3.32 can be applied to find the solution for $\boldsymbol{w}^{(r)}$ as

$$\widetilde{\boldsymbol{A}} = \boldsymbol{A}\left(\boldsymbol{A}^\top \left(\operatorname{diag}\left\{\boldsymbol{c}^{(r)}\right\}\boldsymbol{A}\right)\right)^\dagger \tag{4.11}$$

$$\boldsymbol{w}^r = \left(\widetilde{\boldsymbol{A}}^\top \operatorname{diag}\left\{\boldsymbol{c}^{(r)}\right\}\right)\boldsymbol{l}. \tag{4.12}$$

From the 6-dimensional vector of weights, we save only the w_1, w_4 and w_6 components,

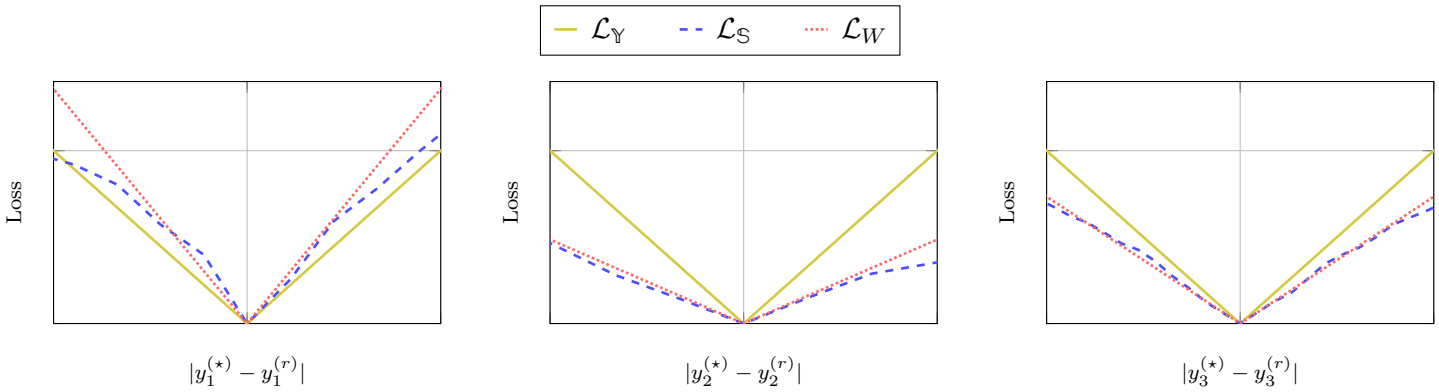


Figure 4.3: Comparison of parameter, spectrum and weighted losses with respect to a reference point $\boldsymbol{y}^{(r)}$. We plot the loss functions $\mathcal{L}_{\mathbb{Y}}(\boldsymbol{y}^{(r)}, \cdot)$, $\mathcal{L}_{W}(\boldsymbol{y}^{(r)}, \cdot)$ and $\mathcal{L}_{\mathbb{S}}(\boldsymbol{b}^{(r)}, \cdot)$. For each plot i, we consider points such that $y_j^{(\star)} = y_j^{(r)}$ for $j \neq i$. Clearly, all the loss functions have minimum in $\boldsymbol{y}^{(r)} = \boldsymbol{y}^{(\star)}$. However, we can observe that the weighted loss $\mathcal{L}_{W}$ approximates better the spectrum loss $\mathcal{L}_{\mathbb{S}}$ in the neighbourhood of the minimum than the parameters loss $\mathcal{L}_{\mathbb{Y}}$.

i.e. coefficients of the second-order pure terms. With the saved coefficients we implement the weighted loss $\mathcal{L}_{W}$ point estimation in the deep learning code. Moreover, we take the square root of the entire loss. Both adjustments aims at making the loss closer to $\mathcal{L}_{\mathbb{S}}$, as well as at increasing performances during the training. Indeed, the limited amounts of coefficients accelerates the training convergence. The square root keeps values distant from the minimum with lower magnitudes. Although the result of approximation is manually tweaked, the approximation is still qualitatively precise in the neighbourhood of the minimum, as we can see in Figure 4.3.

4.3. Models

Two DL models are proposed, i.e, a fully connected neural network, and a deeper convolutional neural network, denoted FC and CNN models respectively. The following sections describe their architectures and their required data processing. More details about training and hyperparameter tuning can be found in Chapter 5.

4.3.1. Fully-connected neural network

The FC model processes the same binned areas $\boldsymbol{b} \in \mathbb{R}^{60}$ adopted in the solution assessment. These areas are normalized and transformed in decibel through $\mathcal{B}_{n_C}^{I_C} \circ \mathcal{N}$. Additionally, the peak value of the spectrum before normalization $\max(\boldsymbol{a})$ is appended to the feature

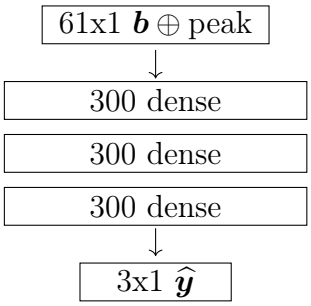


Figure 4.4: Architecture of the FC model. The block "N dense" denotes a fully-connected layer with N neurons.

vector. Figure 4.4 illustrates the NN architecture. It consists of three fully connected layers, also known as dense layer, containing 300 neurons each. The ReLU activation function follows each dense layer. Finally, a three-neurons dense layer with linear activation completes the network. Counting weights and biases, the FC model contains 200 103 trainable parameters[3].

4.3.2. Convolutional neural network

The CNN model consists of a neural network with several 1D convolutional layers. This model tries to relax the preprocessing applied to area signals in the FC by several means. Firstly, given the local shared weights of convolutional layers, CNNs can process larger feature vectors than dense layers. Hence, we apply a less coarse Binning $\mathcal{B}_n^{I_\lambda}$ to the input spectrum $\boldsymbol{s}$, i.e. $n = 1024$ bins. Secondly, we do not integrate spectra on a common wavelengths grid, but rather we perform the binning to each spectrum $\boldsymbol{s} = (\boldsymbol{\lambda}, \boldsymbol{\sigma})$ within its own interval $I_\lambda = (\min(\boldsymbol{\lambda}), \max(\boldsymbol{\lambda}))$. Both options lead to larger bin width, that makes the resulting areas visually closer to the raw spectra. Thirdly, we apply no normalization operator $\mathcal{N}$ to the binned spectra. Finally, since the CNN has no common grid, we stack the spectrum wavelength interval range I_λ to the output feature vector of the backbone network. In this way, the CNN should receive hints about the location of the power densities over the wavelengths.

Figure 4.5 depicts the CNN architecture. The overall architecture is inspired by AlexNet [32]. This CNN includes mainly two blocks, the backbone network and the final dense layers. The backbone network consists of a sequence of trainable convolutions and Average polling. We can notice that the convolutional layer kernels are particularly large, e.g. 25, 9 and 16, with respect to popular CNN architectures, e.g. 3 or 5. In our case, we motivates larger kernels by the fact that data are only in 1D, hence, wider kernels seem to

[3]FC number of trainable parameters: $(61+1)\times300+(300+1)\times300+(300+1)\times300+(300+1)\times3 = 200\,103$

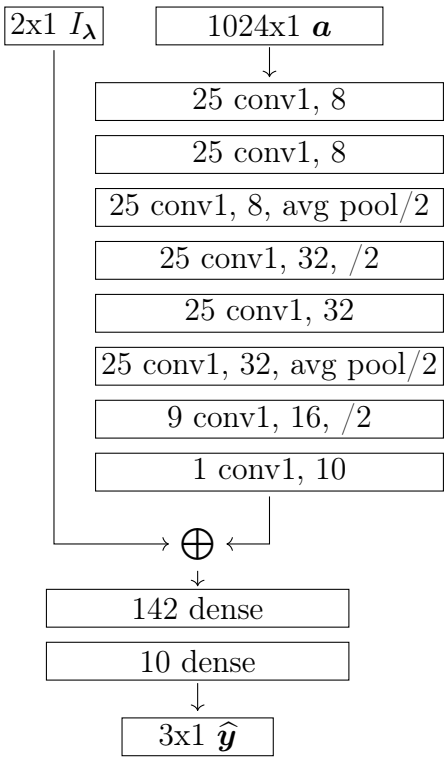


Figure 4.5: Architecture of the CNN model. Each block "K conv1, F" denotes a 1D convolutional layer with kernel width K and number of output channels F. The /2 denotes a stride of 2, when present.

perform best. Average pollings and convolutional layers with stride 2 helps to reduce the dimensionality along the forward passes of the backbone by proving spatial downsampling. A final 1×1 convolutional block ends the backbone by reducing the number of activation maps [35, 55]. Each convolutional block has no bias terms. ReLU activation follows every layer, except for the last output dense layer, where there is a linear activation. Compared to the FC model, the CNN has fewer trainable parameters, i.e. 96 057 coefficients[4].

4.4. Summary

We propose a novel method to solve the Supercontinuum inverse problem. In this method, we train two neural networks, namely FC and CNN, to predict laser pump parameters from preprocessed spectra, i.e. binned areas. In contrast to isotropic parameter loss, both NNs optimize a weighted loss function that better approximates the intractable spectrum loss.

[4]CNN number of trainable parameters: $25 \times 1 \times 8 + 2 \times 25 \times 8 \times 8 + 25 \times 8 \times 32 + 2 \times 25 \times 32 \times 32 + 9 \times 32 \times 16 + 1 \times 16 \times 10 + (200 + 2 + 1) \times 142 + (142 + 1) \times 10 + (10 + 1) \times 3 = 96\,057$

5 | Implementation details

This chapter discusses practical aspects of this research. Firstly, we describe the hardware and software infrastructure. Secondly, we provide more information regarding data generation, e.g. timing and numerical settings. Finally, the chapter describes the hyperparameter tuning processes (HPT) and the space of hyperparameters explored.

5.1. Software and Infrastructure

We implemented the algorithms for data preprocessing and weighted loss approximation in Matlab, and they can comfortably run on modest consumer laptops.

We coded the DL models and training procedures in Python by adopting the open-source API TensorFlow [1] and Keras [8]. Although we could train single models on CPU in a reasonable time, we accelerated the HPT training procedures on graphics cards.

We conducted the experiments in the Tampere University TCSC high-performance computing (HPC) cluster. The cluster orchestrates submitted jobs through Slurm Workload Manager [49]. The available hardware in the cluster includes 140 CPU-only nodes with a total of 3000+ CPU cores and 22 nodes with 4 GPUs in each with different memory sizes. We adopted the same infrastructure to generate the supercontinuum spectra.

5.2. Dataset generation

This work adopted simulated spectra by numerically integrating the GNLSE, i.e. the forward model. The simulator consists of an existing Matlab script that implements the SC generation $\mathcal{G}$ described in section 2.1. We generated a dataset that includes a total of 51429 spectra. Then, for any experiment, we regenerated the spectra from the predicted parameters. The simulator outputs the signal power densities sampled over a non-regular grid of wavelengths which has a constant step in the frequency domain.

We created the set of parameters $\mathbb{Y}$ by the Cartesian product of three linear spaces, one for each pulse parameter to predict. Table 5.1 reports the pulse parameter ranges adopted. Moreover, other SC generation settings are constant since they represent some structural parameters that normally are not trivially adjustable in physical SC lasers. For

	Parameter	Minimum	Maximum	Step	#
λ_0	Pulse wavelength	1400 nm	1700 nm	10 nm	31
τ	Pulse duration (TFWHM)	50 fs	250 fs	10 fs	21
Pp	Peak power	500 W	20 000 W	250 W	79

Table 5.1: Ranges of the pulse parameter linear spaces.

Structural parameter	Value
Time grid points	2^{15}
Time window	140 ps
Pulse shape	hyperbolic secant
Fiber length	6 m

Table 5.2: Structural constant parameters during the SC generation.

the sake of reproducibility, table 5.2 reports the structural parameters.

The simulation is a costly process. As a matter of fact, the generation of all 51429 takes approximately 10 h when runs distributed on a Slurm cluster of 512 nodes. The generation of a single spectrum is not constant in time and depends on the parameter triplet passed. For instance, Figure 5.1 shows that increasing the peak power leads to a higher processing time for the simulation of the forward model. Some simulations can even reach 1 h in time.

5.3. Hyperparameter tuning

We adopted the platform *Weights & Biases* (WandB) [6] to perform Hyperparameter tuning of both FC and CNN models. We employed mainly random search strategies over the slice of hyperparameter space of interests. Bayesian search methods based on Gaussian Processes [5] in practice gave worse results than random searches for our problem.

We parametrized every model configuration, and we associated each hyperparameter of interest with a statistical distribution. Then we pass a YAML configuration of the selected parameters to the WandB platform. Listing 5.1 exemplifies a possible file. We start multiple WandB agents within the HPC cluster. The agents connect to the WanB backend and listen for a set of hyperparameters. Once they receive the hyperparameters, one of our scripts instantiates a model out of those hyperparameters and starts the training. The training continues until the Early Stopping technique activates [43]. During the training iterations, we measure the score E_W, i.e. based on $\mathcal{L}_W$, on a validation set. The script

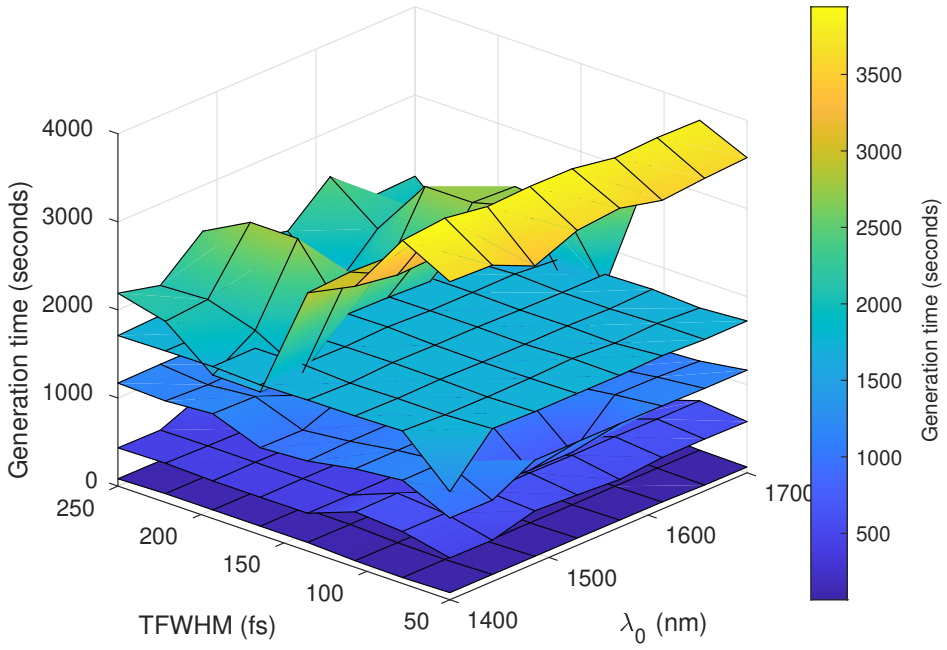


Figure 5.1: Time required by generating spectra given the Power peak. Each surface shows the generation time as function of pulse wavelength λ_0 and pulse duration τ. Every different surface contains points with the same value of power peak Pp. Higher surfaces have higher Pp and on average longer generation time. We can observe that with lower Pp, the generation time is nearly constant with respect to λ_0 and τ. On the other hand, higher values of Pp cause the shorter durations to increase the generation time.

periodically forwards the score to the WandB backend at end of every training epoch. We remark that we cannot measure the spectrum score within the training procedures. The process continues until interruption from the user. WandB UI presents the scores in a convenient web dashboard. From there, we can conveniently assess the impact of every hyperparameter. Finally, we can select the best hyperparameters.

Listing 5.1: Example of WandB HPT configuration file

```
method: random
metric:
  goal: minimize
  name: validation_error
name: CNN arch 10F
parameters:
  batch_size:
```

```
    distribution: categorical
    values: [64, 128, 256, 512]
  learning_rate:
    distribution: log_uniform
    max: -2.3
    min: -11.3
  optimizer:
    distribution: categorical
    values: [adam, sgd]
  cnn_use_bias:
    distribution: categorical
    values: [true, false]
  use_batch_norm:
    distribution: categorical
    values: [true, false]
  [...]
program: train.py
```

The hyperparameter optimized includes the learning rate, different learning rate decay policies, different optimizers like Adam, SGD and RMSProp, the optimizer hyperparameters, the network architecture like number of neurons, number of layers, number of kernels and number of channels, regularization techniques like dropout and batch normalization.

6 | Evaluation

This chapter focuses on the evaluation of the DL models over testing datasets and the impact of the weighted loss function compared to the loss function over the parameters. Firstly, FC and CNN are trained and compared to a nearest neighbour model. Secondly, we explore different preprocessing variations for the CNN model, compared to the preprocessing suggested by experts. Both experiments compare models trained with different training set sizes.

6.1. Experimentation scheme

As Figure 6.1 shows, we randomly shuffle and divide the training set TR into multiple F folds. Then, we train F models separately, where the i-th model $\mathcal{M}_i$ receives training on the i-th fold and testing on the $((i+1) \bmod F)$-th fold, for $i = 1, 2, \dots F$. Hence, the denomination of training set and testing set depends on whether the model has seen the samples during the training or not. Thus, the training set of model $\mathcal{M}_{(i+1) \bmod F}$ coincides with the testing set of model $\mathcal{M}_i$.

The evaluation over fold j gives as output the estimated parameters $\widehat{\boldsymbol{Y}^{(j)}}$ and the error over parameter predictions $E_{\boldsymbol{Y}^{(i)}}$. We obtain the overall parameter error $E_{\boldsymbol{Y}}$ by averaging each parameter error. We run again the simulator over the estimated parameters $\widehat{\boldsymbol{Y}^{(j)}}$ to obtain the generated spectra $\{(\widehat{\lambda}, \widehat{\sigma})\}_i$. Finally we can compute the error over spectra E_S from the newly generated spectra and their ground truths.

In practice, each model $\mathcal{M}_i$ represents the ensemble $\overline{\mathcal{M}}_i$ of $T = 10$ different models $\{\mathcal{M}_i^{(t)}\}_1^T$. We train the models of each ensemble with different weights initializations. Their estimated parameters of a given spectrum $\boldsymbol{s}$ are eventually averaged, such as

$$\widehat{\boldsymbol{y}} = \overline{\mathcal{M}}_i(\boldsymbol{s}) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{M}_i^{(t)}(\boldsymbol{s}). \tag{6.1}$$

This procedure helps reduce overfitting and improves the metric scores [7].

To assess the quality of the models predictions with respect to the size of the training set, we repeat the experiments for different numbers of folds $F = 5, 10, 20$, i.e. training sets of 10285, 5142 and 2571 samples.

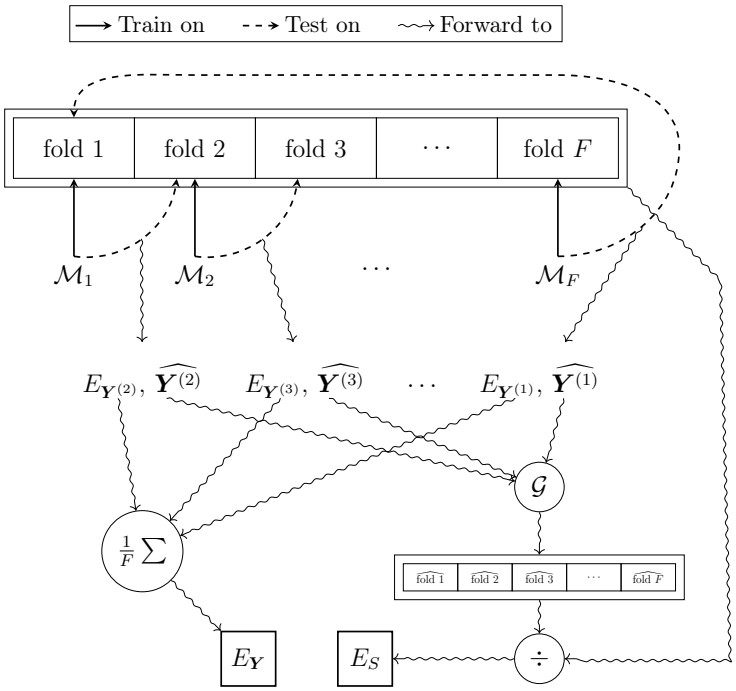


Figure 6.1: Experimentation scheme: training & testing. The schema illustrates how to compute the final metrics $E_{\mathbb{Y}}$ and $E_{\mathbb{S}}$ from a dataset uniformly and randomly split in $\{\widehat{\text{fold i}}\}_{i=1}^{F}$. The symbol $\mathcal{M}_i$ denotes a model trained on a fold. By testing on another fold, each model estimates the fold parameters $\boldsymbol{Y}$. We can calculate the fold parameter errors from the ground-truth parameters $E_{\boldsymbol{Y}}$. Hence, $E_{\mathbb{Y}}$ denotes the aggregation of the fold parameter errors. We forward the fold parameters to the generator $\mathcal{G}$. Finally, we compare $(\div)$ the new generated spectra $\{\widehat{\text{fold i}}\}_{i=1}^{F}$ with the original folds.

6.2. Testset

This experiment aims to test the performances of neural networks over testing SC generated spectra. In this experiment, we apply the experimentation scheme to all the models over the dataset of generated spectra as explained in Section 6.1. Both FC and CNN models separately optimizes the parameter loss function $\mathcal{L}_{\mathbb{Y}}$ and the weighted loss function $\mathcal{L}_W$. Also, we compare the neural networks to the 1-nearest neighbour model, denoted as *1N*. The *1N* model associates to a target spectrum $\boldsymbol{s}^\star$ the parameters $\widehat{\boldsymbol{y}^\star}$ of the closest spectrum in its support $\mathbb{T}$, where we define closeness by the loss function $\mathcal{L}_{\mathbb{S}}$ as in

$$\widehat{\boldsymbol{y}^\star} = \underset{(\boldsymbol{s};\boldsymbol{y})\in\mathbb{T}}{\arg\min}\, \mathcal{L}_{\mathbb{S}}\left[\mathcal{N}(\mathcal{B}_r^R(\boldsymbol{s})), \mathcal{N}(\mathcal{B}_r^R(\boldsymbol{s}^\star))\right], \tag{6.2}$$

where the support $\mathbb{T}$ is the fold on which we are training the *1N* model.

		$F = 5$			$F = 10$			$F = 20$		
		$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$\Delta E_{\mathbb{S}}$
1N		0.123	0.354	-	0.161	0.449	-	0.206	0.569	-
FC	$\mathcal{L}_{\mathbb{Y}}$	0.029	0.168	(-7.1%)	0.045	0.235	(-8.5%)	0.066	0.329	(-5.5%)
	$\mathcal{L}_W$	0.032	0.156		0.047	0.215		0.070	0.311	
CNN	$\mathcal{L}_{\mathbb{Y}}$	0.014	0.130	(-11.5%)	0.022	0.186	(-7.5%)	0.034	0.268	(+1.5%)
	$\mathcal{L}_W$	0.017	0.115		0.028	0.172		0.044	0.272	

Table 6.1: Results of model predictions over the testing generated spectra. Each row consider a different model, namely 1-nearest neighbour *1N*, fully-connected FC and convolutional neural network CNN. We trained the neural networks both with parameter loss $\mathcal{L}_{\mathbb{Y}}$ and weighted loss $\mathcal{L}_W$. We repeated each training-test procedure for the dataset split in $F = 15, 10, 20$ folds.

6.2.1. Results

Table 6.1 reports the resulting metrics computed for this experiment.

Firstly, the reader will notice that the neural networks perform better than the *1N* model. This is certainly beneficial for memory complexity, as it is then possible to achieve better performances without memorizing all the spectra in the support of the *1N*. In particular, the CNN model outperforms the FC in every instance of this experiment. This could be due to the fact that convolutional layers learn a better representation than to the heavier preprocessing applied to the spectra fed to the FC. Secondly, as it was easy to forecast, the performances of all the models deteriorate when the fold number F increases, i.e. the number of samples in the training set decreases.

Finally, the optimization over $\mathcal{L}_W$ helps to achieve better similarities between target spectra and spectra generated from the estimated parameters, compared to optimizations over $\mathcal{L}_{\mathbb{Y}}$. We can notice that for $F = 20$, the weighted loss function loses its efficacies for the CNN model. We speculate that the trainset at $F = 20$ is too sparse for the weight computation to bring benefits to the CNN model. Additionally, the parameter error $E_{\mathbb{Y}}$ increases in trainings with the weighted loss function. This result shows that getting closer to the minimum of the spectrum error does not imply getting closer to the minimum in the parameter space.

6.2.2. Visualization

Figure 6.2 reports some examples of spectra generated from the predicted parameters of the neural networks. As we can see, both FC and CNN are able to predict the parameters such that generate spectra close to the target ones. In particular, for predictions with lower spectra error $E_{\mathbb{S}}$, e.g. within the FC-best 60% predictions, both spectra profile and relative areas almost overlap totally with the desired targets.

From the third row of Figure 6.2, the target spectra with predictions with higher error seem to still lead to reasonable generated spectra. From these examples, we can notice that the CNN performs better, and the FC tends to predict parameters that generate slightly shifted spectra in the wavelengths.

6.3. Testing different CNN preprocessings

This experiment aims at understanding the best preprocessing pipeline to apply to the CNN input spectra. The assumption is that the CNN model requires no preprocessing, and we should forward the raw spectra to the CNN. By adopting the same experimentation scheme defined in Section 6.1, we test several combinations. We combine plain binning vs. common-grid binning with FC-like normalization vs. no normalization. Also, we test void preprocessing operator, i.e. we feed plain filtered spectra given by the generator.

We always test spectra in $\mathbb{R}^{1024}$. In the case of no binning, we apply a decimator to filter the original output of the generator, i.e. spectra in $\mathbb{R}^{4096}$. In the case of binning, we train on 1024-bins areas. When we employ no common grid in the binning operator, we append the wavelength interval to the input vector, i.e. +2 in its dimensionality. When we apply the FC-like dB-normalization, we clip and stack the maximum peak of the power density profile, so +1 in the input dimensionality.

Within the CNN setup, We always apply scalers to input vectors after possible normalization, as they improve scores and performances of the training procedure. When we apply dB-normalization, CNN uses a standard scaler computed feature-wise. Otherwise, without dB-normalization, the global min-max scaler has better performances.

Table 6.2 reports the results of the experiment. We can notice that the dB-normalization does not bring better benefits to the CNN model. Additionally, binning the spectra over the same interval, i.e. common-grid binning, is not better than passing the wavelength range I_λ over which the input spectrum has been integrated. This means that the CNN can leverage the information about spectrum wavelengths.

Binning adoption is still convenient. Contrarily to what we expected, the raw spectra perform similarly to the binning alone, but not better. This could be due to the choice of

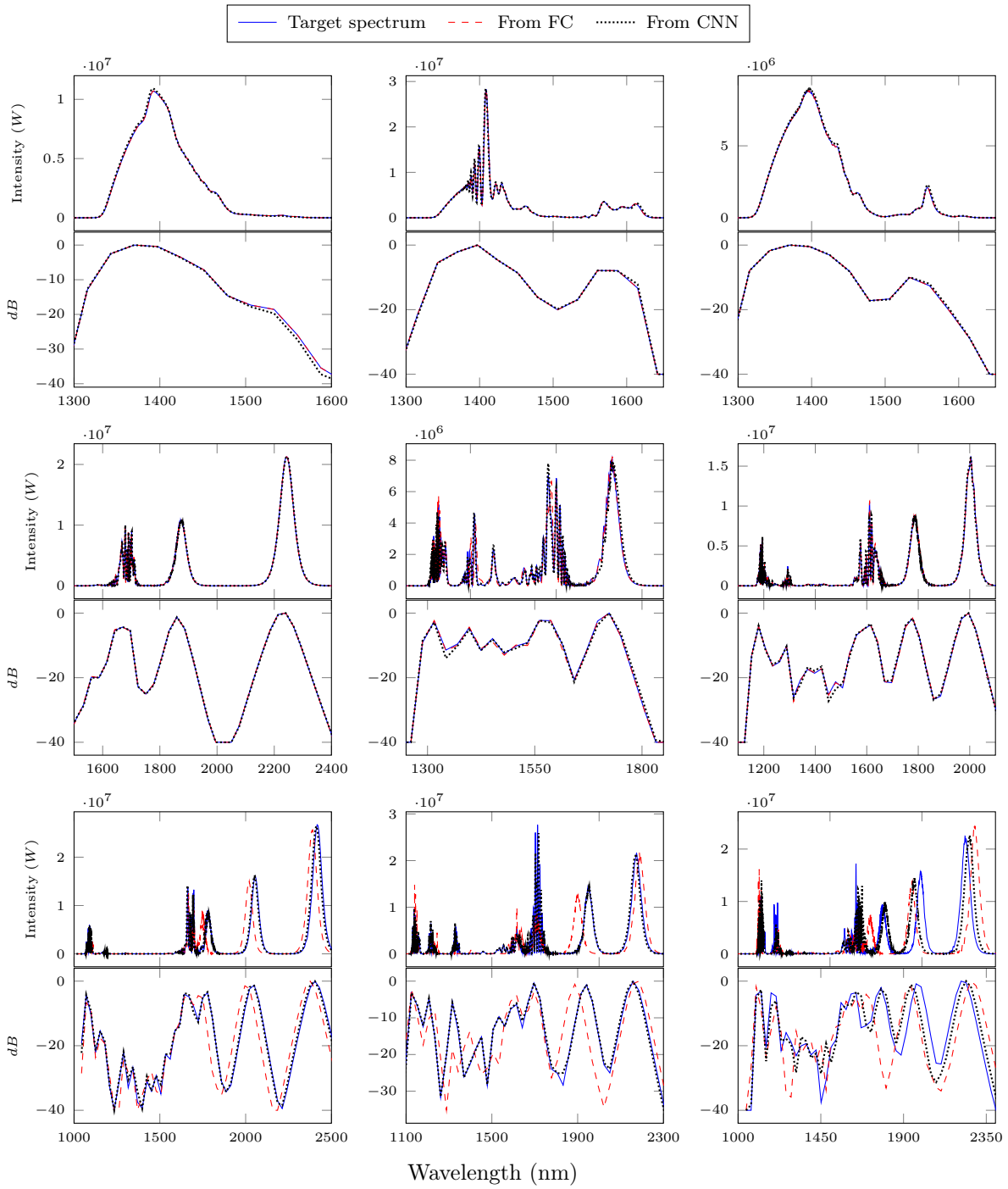


Figure 6.2: Spectra and relative areas generated from testing set predictions of neural networks. Each row contains a set of 3 spectra, with their relative binned areas showed underneath. The spectra are randomly sampled from the FC $E_{\mathbb{S}}$ error intervals of $[0; 10]$-percentiles, $[40; 60]$-percentiles and $[90; 100]$-percentiles for the first, second and third row respectively.

		$F = 5$		$F = 10$		$F = 20$	
		$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}$
Common-grid Binning, dB-norm	$\mathcal{L}_{\mathbb{Y}}$	0.029	0.245	0.050	0.339	0.065	0.471
	$\mathcal{L}_{W}$	0.034	0.231	0.052	0.330	0.085	0.474
Common-grid Binning	$\mathcal{L}_{\mathbb{Y}}$	0.018	0.140	0.024	0.209	0.038	0.300
	$\mathcal{L}_{W}$	0.021	0.123	0.028	0.197	0.044	0.304
Binning, dB-norm	$\mathcal{L}_{\mathbb{Y}}$	0.022	0.198	0.040	0.305	0.062	0.378
	$\mathcal{L}_{W}$	0.025	0.182	0.042	0.273	0.067	0.373
Binning	$\mathcal{L}_{\mathbb{Y}}$	0.014	0.130	0.022	0.186	0.034	0.268
	$\mathcal{L}_{W}$	0.017	**0.115**	0.028	**0.172**	0.044	**0.272**
Raw spectra	$\mathcal{L}_{\mathbb{Y}}$	0.013	0.125	0.026	0.185	0.042	0.273
	$\mathcal{L}_{W}$	0.018	0.119	0.030	0.183	0.045	0.279

Table 6.2: Results of different CNN preprocessings. Each row represents the results over the testing set of two model trained with parameter loss $\mathcal{L}_{\mathbb{Y}}$ and weighted loss $\mathcal{L}_{W}$, respectively. Each row follows a different preprocessing combination, as explained in Section 6.3. We highlight the result ranking of $E_{\mathbb{S}}$ for the model trained with $\mathcal{L}_{W}$ as best in **dark green**, second-best in light green, second-to-worst in purple and worst in red.

the architecture, since we optimize the hyperparameters only with respect to the 1024-binned spectrum inputs. A separate HPT for each row of this table is too expensive. Future deeper investigation of better architectures could unlock more benefits for raw spectra.

6.4. Summary

Both experiments shows that the weighted loss function improves the spectra error in the testing set, i.e. the generated spectra from network predictions are closer to target spectra than when the standard MAE over parameters is used. Both neural networks perform better than 1-nearest-neighbour searches over the training set. In particular, the CNN outperforms the FC, due to the higher dimensionality retained during the binning integration. The last experiment aimed at determine the best spectrum preprocessing for the CNN model. This confirms that the expert-driven preprocessing applied in the FC discard information and cannot be applied in the CNN without losing margin in the spectra error.

7 | Adaptation to non-SC-generated spectra

The ultimate goal of this project is to provide a solution to scientists and practitioners, i.e. users, that explains the SC laser pump parameters to generate the desired target spectra. To express the desired target spectra, users must provide a vector representing the spectrum profiles. The representation given by users may not replicate the nonlinearities typical of the spectra provided by the generator with which we train the NNs. Therefore, it is crucial to test the generalization of the FC and CNN over non-generated spectra.

This chapter shows the current limitation of our solution to real-world spectrum profiles and reports further experiments that test models over spectra not provided by the generator. In particular, we analyse two examples, Gaussian spectra and absorption spectra. Finally, this chapter provides hints about adjustments to the solution to improve the results.

From a user perspective, Gaussian spectra represent the scenario where persons could hand-sketch a spectrum to centre wavelength ranges of interest. Gaussian spectra are rougher than SC generated ones, so they are not so precise. On the other hand, absorption spectra represent the industrial scenario where a target spectrum is well-known. Often, the absorption spectra are even more detailed than SC generated spectra.

7.1. Gaussian spectra

Section 6.2 describes the evaluation of the proposed solution over spectra from the generator $\mathcal{G}$. However, it is natural to wonder about the generalization performances of the models over spectra that resembles the generated ones, but that are not coming from the generator. In this section we tried the neural networks over two datasets of Gaussian spectra. The first one contains spectra obtained by fitting Gaussian mixture models (GMM) over the testing SC generated spectra. The second dataset contains synthetic spectra defined as sum of random Gaussian curves (SRG). Both tests aim to evaluate the generalization abilities of the models over spectra that are smooth enough to resemble human-drawn profiles, i.e. spectrum profiles that would not contain high-frequency nonlinearities typical

of generated spectra.

First, we compute the GMM dataset by fitting 1 to 5 mixture components to the testing set spectra. After filtering anomalies and degenerate profiles, this resulted in approximately 45000 spectra, whose areas flow for evaluation to the FC and CNN model trained optimizing the weighted loss function $\mathcal{L}_W$. We collect and feed the parameters predicted by each ensemble model $\mathcal{M}_i$ to the generator, as it happens similarly in the testing set experimentation scheme described in Section 6.1.

Second, the SRG dataset challenges the NN models even more, since samples are not Gaussian fitted over the testset, but sum of randomly constructed Gaussian curves. All these spectra live in the same wavelength grid $\boldsymbol{\lambda} \in \mathbb{R}^{4096}$, constant in the frequency domain, within the interval $(\lambda_{min}, \lambda_{max})$, where $\lambda_{min} = 905\,\text{nm}$ and $\lambda_{max} = 3976\,\text{nm}$. We sample the power density $\boldsymbol{\sigma}$ of each SRG record in the wavelengths $\boldsymbol{\lambda}$ from a random function $s(\lambda)$ defined as

$$s(\lambda) = \sum_{c=1}^{C} a_c \cdot \exp\left(-\frac{(\lambda - \mu_c)^2}{2d_c^2}\right), \tag{7.1}$$

where C is the desired number of random Gaussian curves, i.e. components, $a_c \sim \mathcal{U}(10^5, 10^8)$ is the component amplitude sampled from a uniform distribution $\mathcal{U}$, $d_c \sim \mathcal{U}(10, 70)$ is the component distortion, $\mu_c \sim \mathcal{U}(\lambda_{min} + 2.8 \times d_c, \lambda_{max} - 2.8 \times d_c)$ is the component mean. For samples with $C > 1$, we constraint the components to not overlap by resampling until $\mu_i \notin (\mu_j - 2.5 \times d_j, \mu_j + 2.5 \times d_j)$, $\forall i, j = 1 \dots C \wedge i \neq j$. The dataset totally contains 900 spectra, i.e. 300 samples for each chosen $C = 1, 2, 3$. We manually tweaked all the aforementioned constants to obtain spectra that visually resemble the macroscopic features of generated SC spectra.

In both experiments, the ground-truth spectra parameters $\boldsymbol{y} \in \mathbb{Y}$ are not available. Their parameters are unknown, as these spectra do not come from the generator $\mathcal{G}$. Therefore, we cannot compute the parameter error. Additionally, the accuracy of the models is much lower, compared to the testing set spectra, as expected. Hence, we reported no spectrum error, as the metric is not expressive for spectra generated from predictions that differ substantially from the corresponding input areas. Nevertheless, we visually assess the results.

Figure 7.1 illustrates results from predictions over the GSM dataset. We can notice that the spectra generated from CNN and FC predictions do not overlap like in the testing set experiment (section 6.2). However, the networks still estimate reasonable parameters. This is true also for worse predictions, i.e. predictions with high spectrum error.

On the other hand, the SRG dataset is more challenging because these Gaussian spectra and areas are not close to any input seen by the network. Figure 7.2 reports three

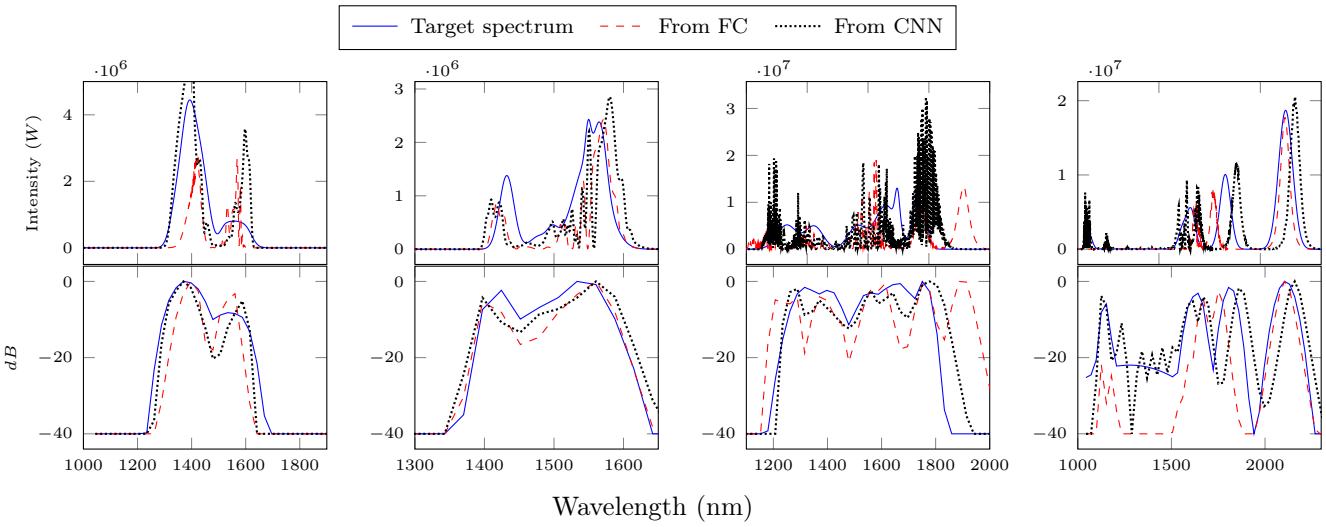


Figure 7.1: Spectra (above) and relative areas (below) generated from predictions over the Gaussian mixture model dataset (GMM). Each column shows a spectrum sampled from the FC $E_{\mathbb{S}}$ error intervals of $[0; 25]$-percentiles, $[26; 50]$-percentiles, $[51; 75]$-percentiles and $[76 : 100]$-percentiles, respectively.

examples of SRG areas and the corresponding predictions from a FC model. While the network performs reasonably on some cases (Figure 7.2 left and centre), on average do not provide yet acceptable results (Figure 7.2). Indeed, if we order the SRG samples by spectrum error $E_{\mathbb{S}}$, from the 40-percentile to the worst one we obtain results similar to the right side of Figure 7.2.

7.2. Real-world spectra issues

In this section, we discuss two issues related to the numerical scale of spectra. Firstly, the actual scale of energy emitted by the spectra might be not available in real-world applications. Secondly, the dB scale might trick models during the learning phase. These issues affect the evaluation of the current proposed solution. However, Section 7.3 shows that our method is still valid when we transform the testing set to overcome these issues.

7.2.1. Spectrum peak not always available

The maximum peak of spectrum profiles, thus of the binned areas, is a crucial piece of information required by the FC model to predict parameters, especially the power peak. The preprocessing applied in the FC model normalizes the binned areas, so the energy intensity information would be lost if we do not append the spectrum peak to the feature

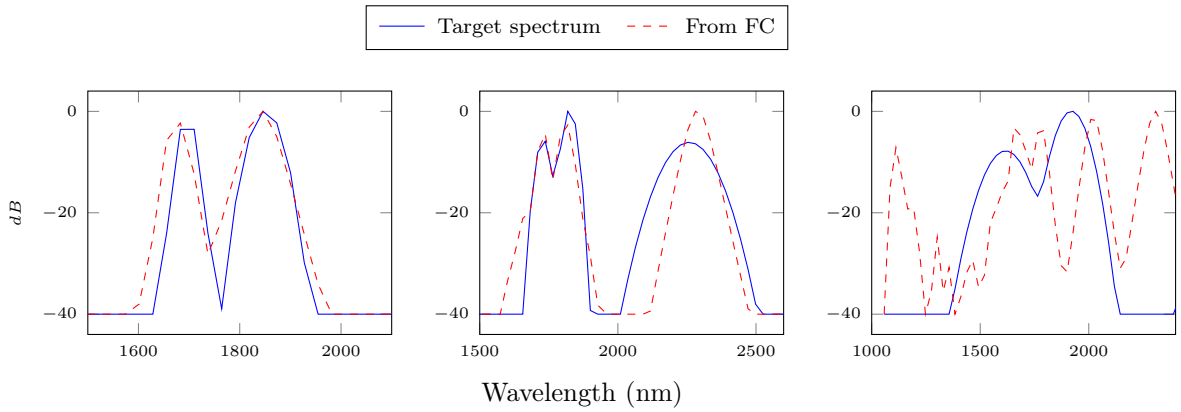


Figure 7.2: Spectra and relative areas generated from Gaussian dataset predictions (SRG). Each column shows a binned areas sampled from the FC $E_{\mathbb{S}}$ error intervals of $[0; 20]$-percentiles, $[21; 40]$-percentiles, and $[41; 60]$-percentiles, respectively.

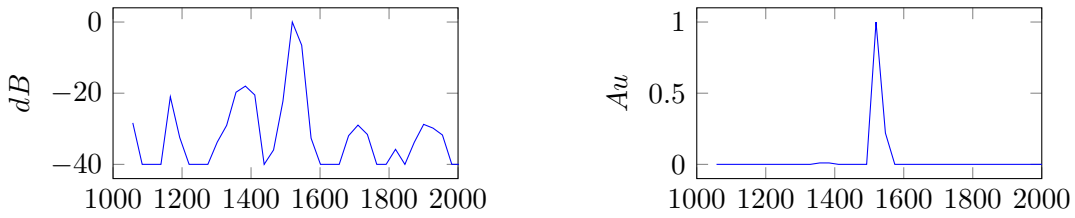
vector. The CNN model still achieves good predictions without stacking of maximum peak value in the feature vector because its spectra are not normalized. So, the peak information is still present.

However, many scientific domains, such as spectroscopy and electronics, often employ normalized or arbitrary units (Au) to express intensities. For instance, this is the case for absorption spectra shown in Figure 7.5. The arbitrary scale can replace scales that are dependent on experimental conditions [27], which serve no purpose in comparing quantities obtained by different experimental equipment. If not all the experimental conditions are known and reproducible, arbitrary units are required. Nevertheless, Au still permits comparing samples acquired in similar settings, e.g. the intensity of peaks at different wavelength points in the spectrum.

Thus, we need to train both models without peak information, in order to adopt them to predict parameters of real-world spectra. This will impact the score over the parameter error, but eventually, we are interested in minimizing the spectra error and obtaining a similar profile to the target one. The spectra magnitude indeed influences the power peak parameter, so the prediction of the power peak given no peak information to relate to is a challenging task. However, it is still possible to solve that, as the shape and distribution of the spectrum profile over the wavelength domain also influence the power peak parameter.

7.2.2. Decibel scale issues

Scientists and engineers widely adopt dB units to process and visualize signals. Similarly, in our case, domain experts recommend the dB scale to compare spectral areas. Therefore, the $\mathcal{N}$ operator defined in Equation (4.5) performs the conversion to the dB scale. We



(a) dB Normalization (b) Arbitrary units

Figure 7.3: Comparison of areas in dB units vs Au. Decibel units amplify weak peaks that Arbitrary linear units would not perceive.

apply this operator both to preprocess the input binned areas of the FC model and the areas regenerated from FC and CNN predictions. Despite the dB unit seems reasonable in the assessment, and also necessary for the FC to get good performances, it might introduce problems when testing the generalization of the FC over spectra non generated from $\mathcal{G}$. As a matter of fact, predicting parameters for spectra not generated by $\mathcal{G}$ is already problematic. Sometimes, generating spectra similar to the target ones with SC lasers might be even physically impossible. For this reason, it is essential to match at least the major energy components in the spectra profile.

Figure 7.3 highlights the problem. The dB units amplify weaker peaks since Decibels constitute a logarithmic scale. The real issue consists in the fact that we linearly compare transformed area values. This comparison will see identical errors of, for example, -20 dB regardless that they happen around at -20 dB or 0 dB. Consequentially, the peak amplifications could trick both the FC model and the evaluation. On the contrary, adopting linear units, such as in plain binned areas without Normalization operator $\mathcal{N}$ or Au, could push the FC in a better direction.

In the following section, we assess the performance of FC and CNN when trained and tested on Au spectra.

7.3. Other surrogate losses over linear Au spectra

The goal of this experiment is to verify whether the weighted loss function method increases the performances over the testset with Au-scaled spectra. Moreover, we experiment with multiple loss function definitions in addition to the MAE of the binned areas, i.e. $\mathcal{L}_{\mathbb{S}}$ (4.6) adopted in Section 4.2.1. We introduce the cosine spectral loss defined as

$$\mathcal{L}_{\mathbb{S}}^{CO}(\boldsymbol{a}, \widehat{\boldsymbol{a}}) = 1 - \frac{\langle \boldsymbol{a}, \widehat{\boldsymbol{a}}^{\top} \rangle}{\|\boldsymbol{a}\| \cdot \|\widehat{\boldsymbol{a}}\|}. \tag{7.2}$$

Model	Loss	$E_{\mathbb{Y}}$	$E_{\mathbb{S}}^{CO}$	$\Delta E_{\mathbb{S}}^{CO}$	$E_{\mathbb{S}}^{MAE}$	$\Delta E_{\mathbb{S}}^{MAE}$	$E_{\mathbb{S}}^{IOU}$	$\Delta E_{\mathbb{S}}^{IOU}$
FC	$\mathcal{L}_Y$	0.068	0.0164	—	0.0088	—	0.1035	—
	$\mathcal{L}_W^{CO}$	0.101	0.0165	+0.84%	0.0092	+4.98%	0.1094	+5.74%
	$\mathcal{L}_W^{MAE}$	0.070	**0.0155**	−5.33%	0.0087	−1.04%	**0.1001**	−3.28%
	$\mathcal{L}_W^{IOU}$	0.072	0.0191	+16.84%	**0.0082**	−6.17%	0.1022	−1.28%
CNN	$\mathcal{L}_Y$	0.028	0.0118	—	0.0061	—	0.0789	—
	$\mathcal{L}_W^{CO}$	0.054	0.0114	−3.14%	0.0064	+5.12%	0.0845	+7.13%
	$\mathcal{L}_W^{MAE}$	0.029	**0.0105**	−11.56%	0.0059	−4.09%	**0.0747**	−5.29%
	$\mathcal{L}_W^{IOU}$	0.033	0.0115	−2.94%	**0.0057**	−6.92%	0.0754	−4.39%

Table 7.1: Performances of different surrogate loss functions over 10 folds. Each row shows the results of a neural network trained with a different loss function, i.e. the parameter loss and three weighted loss functions, as explained in Section 7.3. Numbers in **bold** highlight the best score for a particular spectrum error $E_{\mathbb{S}}^{\star}$ in each model. We compute the improvements $\Delta E_{\mathbb{S}}^{\star}$ with respect to the model trained by $\mathcal{L}_{\mathbb{Y}}$.

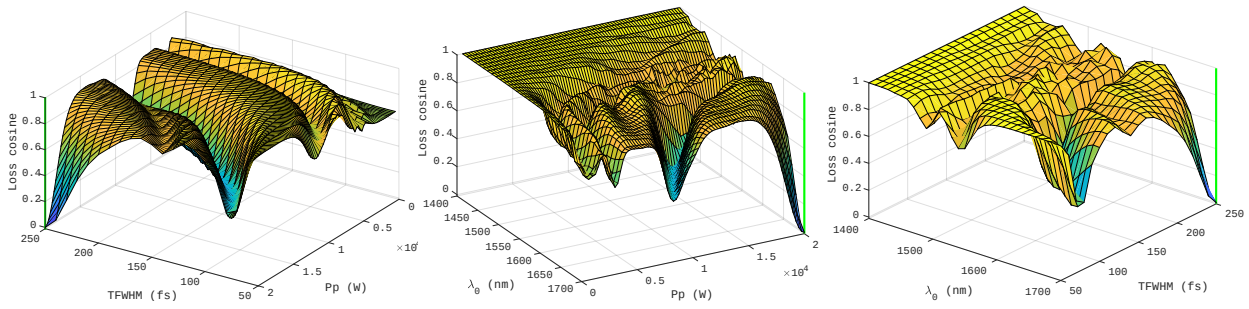
We also consider the intersection-over-union spectral loss defined as

$$\mathcal{L}_{\mathbb{S}}^{IOU}(\boldsymbol{a}, \widehat{\boldsymbol{a}}) = \frac{\sum_i^{n_C} \min\left(a_i, \widehat{a}_i\right)}{\sum_i^{n_C} \max\left(a_i, \widehat{a}_i\right)}, \tag{7.3}$$

where the areas are normalized to have their sum to 1, that is $\sum_i a_i = 1$. Both new loss functions aim to consider the shared energy between the two area signals, rather than the point-wise errors penalized by the plain MAE.

We run the experiment with the same operations of Section 6.1 for each architecture, FC and CNN, with the dataset partitioned into $F = 10$ folds. Similarly to the experiments in Section 6.2 and Section 6.3, we measure the performances on the spectra generated by the neural network predictions. Despite the loss used in the training, we assess each method by metrics $E_{\mathbb{Y}}$, $E_{\mathbb{S}}^{Co}$, $E_{\mathbb{S}}^{MAE}$ and $E_{\mathbb{S}}^{IOU}$, defined respectively by the expression of $\mathcal{L}_{\mathbb{Y}}$, $\mathcal{L}_S^{CO}$, $\mathcal{L}_S^{MAE}$ and $\mathcal{L}_S^{IOU}$.

Table 7.1 reports the resulting metrics of the experiment and the improvements over the baseline model trained with $\mathcal{L}_{\mathbb{Y}}$. Firstly, we can notice that almost every weighted loss function improves at least the corresponding metric. On the contrary, the FC trained with cosine loss function provide no gain for the metric $E_{\mathbb{S}}^{CO}$. The CNN trained with $\mathcal{L}_W^{CO}$ only slightly improves $E_{\mathbb{S}}^{CO}$. However, in both networks, the error over the parameters



(a) Fixed λ_0 (b) Fixed TFWHM (c) Fixed Pp

Figure 7.4: Cosine loss evaluated on a sample $\boldsymbol{y}^\star = \begin{pmatrix} 1700\,\text{nm} & 250\,\text{fs} & 20\,000\,\text{W} \end{pmatrix}^\top$. Each figure $i = 1, 2, 3$ shows the plot of $\mathcal{L}_W^{CO}(\boldsymbol{y}^\star, \boldsymbol{y})$ projected on the subspace $y_i^\star = y_i$. The green vertical line indicates the loss minimum, i.e. $\boldsymbol{y}^\star$.

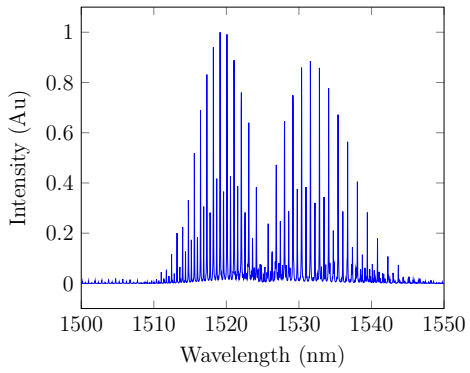
$E_{\mathbb{Y}}$ is higher than the average of other training procedures. As Figure 7.4 might suggest, the reason could lie in the oscillatory behaviour typical of this loss. Neural network optimization could probably stop in minima different from the global one. We hypothesize that this loss might unlock its benefits if used to fine-tune the models. We will verify this in future work.

All the training procedures with $\mathcal{L}_W^{MAE}$ and $\mathcal{L}_W^{IOU}$ achieve good results. The results are not directly comparable with metrics of Section 6.2 and Section 6.3 because of the area normalization in the Au scale. The normalization to Au causes spectral errors to have different scales compared to $E_{\mathbb{S}}$. Moreover, this task is more challenging since the NNs have no reference magnitude to distinguish spectra whose parameters would lead to different energy magnitudes.

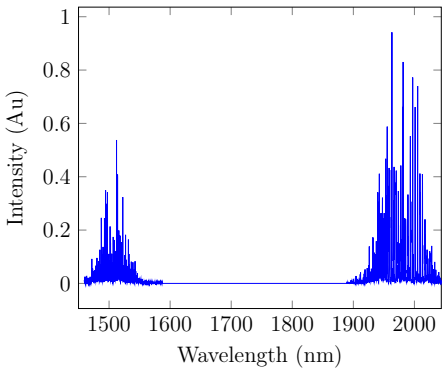
7.4. Testing on absorption spectra

The goal of this experiment is to visually assess the performance of our proposed method over real-world absorption spectra. We can define an absorption spectrum of a given material as the resulting portion of incident radiation that the material has absorbed [26]. The absorbed frequencies depend on the atomic and molecular configuration of the material. This property makes absorption spectra widely employed in many branches of Spectroscopy because it is then possible to recognize substances. For example, in Chemical Spectroscopy, absorption spectra allow detecting the presence of anomalous substances in power plants or pollutants in the air. In Astronomy, scientists employ absorption Spectroscopy to study constellation compositions and planet characteristics.

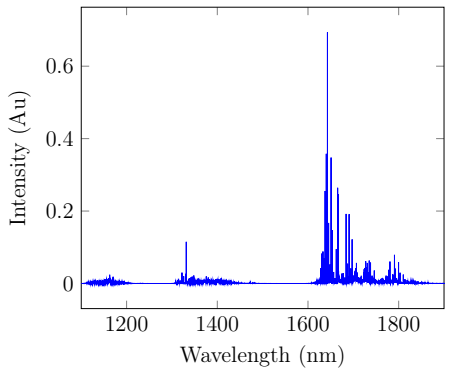
Given a target material, SC generation can help generate the proper broadband



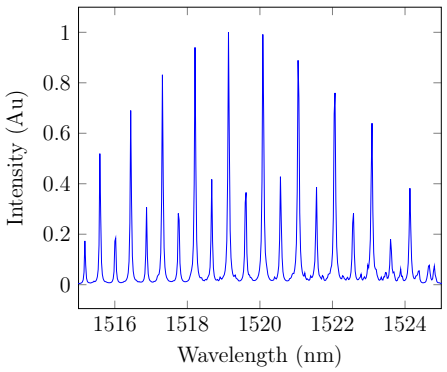
(a) Acetylene absorption spectrum.



(b) Ammonia absorption spectrum.



(c) Methane absorption spectrum.



(d) Detail of Acetylene absorption spectrum in wavelength range $(1515, 1525)$ nm.

Figure 7.5: Absorption spectrum power densities.

spectrum required to cover the bands of the material absorption spectrum. Then, we can use our method to predict the laser pump parameters that would generate a spectrum close to the absorption one. Hence, we consider the absorption spectra use case as an arena for our models. We tested over the absorption spectrum of three gases, i.e. Acetylene, Ammonia and Methane. Figure 7.5 reports their power densities.

Taking Acetylene as an example, Figure 7.5d shows that the intensity of this kind of spectra is highly sparse in wavelengths. Given the problem formulation described in Section 2.2, the prediction of the absorption spectra parameters turns the problem into ill-posed since SC generation cannot produce the absorption spectra. However, by testing our method on these samples, we aim to obtain some spectra that are as close as possible to the envelope of the target absorption.

The experiment considers only the FC model. We want to verify whether there is any difference in training with weighted loss and without. Absorption spectra suffer the issues discussed in Section 7.2.1, so we scaled the areas in Au. Given the results in Section 7.3, the intersection-over-union weighted loss $\mathcal{L}_W^{IOU}$ offers good performances. Therefore, the

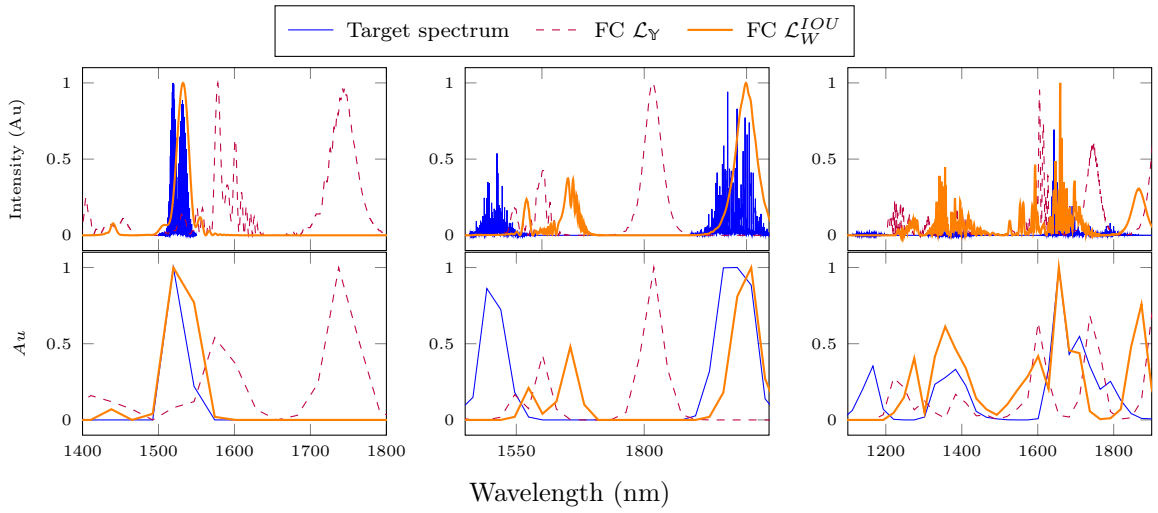


Figure 7.6: FC predictions over absorption spectra. We trained two FC models on $F = 2$, optimizing the parameter loss and the intersection-over-union weighted loss respectively. The first row reports the Au spectra of Acetylene, Ammonia and Methane. The second one illustrates the Au binned areas of the absorption spectra.

weighted version of the FC optimizes $\mathcal{L}_W^{IOU}$.

Figure 7.6 shows the spectra generated by the FC model predictions for the absorption spectra. While we can notice that our method is still far from predicting reasonable values, we can still appreciate the positive impact of the weighted loss function. This is especially clear in the predictions over the Acetylene. In fact, the weighted loss clearly helps at predicting parameters that can centre the corresponding spectrum. Moreover, without weighted loss, the predicted parameters would result in an inappropriate number of macroscopic peaks. On the other hand, the weighted loss function only helps centre the generated spectra for the Ammonia and Methane targets. However, these two absorption spectra present a profile that even in the binned areas is quite different from training set spectra.

7.5. Summary

The proposed solution has the potential to be applied in general scenarios, where the spectra are not SC generated, e.g. Gaussian and Absorption spectra. The results are promising, but the models do not yet fully deliver sufficiently in non-generated spectra. However, the weighted loss function approach is advantageous and help in improving the results. In particular, we show that the weighted loss function method also integrates well to different ideal losses than the MAE over the binned areas (4.6). Moreover, the

same benefits highlighted in Chapter 6 are still present when we deprive the area scale information to the networks. Finally, the weighted loss helps to visually improve the quality of regenerated spectra from the predictions on absorptions.

8 | Conclusions

This thesis proposes a solution to the inverse problem of parameters estimation for the generation of Supercontinuum spectra. This inverse problem consists of obtaining the parameters of initial laser pulses that, when propagated into nonlinear media, e.g., optic fibres, can generate target broadband spectra. A resource-intensive generator program simulates the forward pass of the Supercontinuum generation.

The proposed solution consists of two deep learning models, fully connected and convolutional neural networks, that have learnt the backward pass and can predict the laser pump parameters. Both models can lead to lower errors over the regenerated spectra when compared to the nearest neighbour search. The models are efficient both during training and evaluation since we can employ them without necessary access to GPUs.

In addition, the proposed solution includes a new method for approximating loss functions over inverse problems forward domain. In our case, we employed this method to compute a weighted loss function to approximate the spectrum loss. Therefore, we are not limited to typical isotropic parameter loss functions. We adopted this loss to train the neural networks and overcome the simulator computational intractability.

We found that this method improves testset results in the spectrum domain when compared to models trained without it. Finally, we investigated the generalization performance of our solution over real-world spectra. Although the performance needs to improve, we still found improvements when adopting the weighted loss function.

The proposed weighted loss definition is independent of the specific domain of the Ultrafast Optics. Therefore, it has the potential to improve results in other inverse problems.

8.1. Future work

The goals of future work are two-fold: solve the SC generation inverse problem in the real world, and test the validity of the weighted loss function in other similar inverse problems and against alternative solutions.

To achieve good performances when addressing real-world spectra, like absorption spectra, we will focus on improving the prediction accuracy and robustness, as well as

increasing the generalization of the models. Hence, future work will experiment with modified and novel neural network architectures, e.g., deeper CNN, or Autoencoders to learn the SC generator process. Additionally, we would like to increase the solution space dimensionality, by predicting additional parameters that we have not considered so far, such as the right length of the optical fibre. Finally, we would like to move to real data from physical lasers, in contrast with the current simulated spectra.

We believe that improving the weighted scheme will benefit the SC inverse generation and other similar inverse problems. As Section 7.3 highlighted, the weighted loss can approximate several possible loss definitions. Therefore, finding the most suitable one for our problem is yet an open question. Moreover, we will consider new weighted loss expressions, in contrast with the current one, i.e. Equation (4.8). For instance, we would like to extend our method by defining a noise model over the real loss function to approximate, and employing an anisotropic and spatial adaptive implementation [19], in contrast with our current neighbour-isotropic and non-adaptive method. Concerning this extension, we could adopt the algorithm defined by the concept of local polynomial approximation and intersection of confidence intervals (LPA-ICI) [13].

Regarding the validity of our method, we plan to test the weighted loss function method on other inverse problems. Therefore we can investigate if the positive impact of the weighted loss is reproducible. Among other inverse problems, the optimization of Aereonautics geometries [50] from fluid dynamics features could be a possible arena. In this research, neural networks predict airfoil shape parameters from the preprocessed wind velocity maps. This problem resembles the SC generation, since the generation of wind maps from airfoil shapes is an expensive simulation. Thus, we could easily insert our method to optimize the airfoil parameters that lead to wind maps as close as possible to target ones. Finally, the work in [48] adopted the Nelder-Mead optimization to estimate the violin plate parameters given target vibrational properties. This represents yet another parameter estimation inverse problem where our method could be applied. Therefore, the inverse process could be learnt by a neural network optimized via weighted loss function.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Govind Agrawal. Pulse propagation in fibers. In Govind Agrawal, editor, *Nonlinear Fiber Optics (Fifth Edition)*, Optics and Photonics, pages 27–56. Academic Press, Boston, 2013. doi: https://doi.org/10.1016/B978-0-12-397023-7.00002-4. URL `https://www.sciencedirect.com/science/article/pii/B9780123970237000024`.

[3] Mauricio Araya-Polo, Joseph Jennings, Amir Adler, and Taylor Dahlke. Deep-learning tomography. *The Leading Edge*, 37(1):58–66, 2018.

[4] FR Arteaga-Sierra, C Milián, I Torres-Gómez, M Torres-Cisneros, H Plascencia-Mora, G Moltó, and A Ferrando. Optimization for maximum Raman frequency conversion in supercontinuum sources using genetic algorithms. *Revista Mexicana de Física*, 63 (2):111–116, 2017.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

[7] Christopher M Bishop. *Pattern Recognition and Machine Learning*, chapter 14. Springer, 2006.

[8] François Chollet et al. Keras. `https://keras.io`, 2015.

[9] DeepAI. Max Pooling. `https://deepai.org/machine-learning-glossary-and-terms/max-pooling`. Accessed: 2021-09-27.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[11] John M Dudley, Goëry Genty, and Stéphane Coen. Supercontinuum generation in photonic crystal fiber. *Reviews of Modern Physics*, 78(4):1135, 2006.

[12] Gunnar Farnebäck. A unified framework for bases, frames, subspace bases, and subspace frames. In *Proceedings of the 11th Scandinavian Conference on Image Analysis*, pages 341–349, 1999.

[13] Alessandro Foi. Anisotropic nonparametric image processing: theory, algorithms and applications. *Dip. di Matematica, Politecnico di Milano*, 2005.

[14] Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Pointwise shape-adaptive dct for high-quality denoising and deblocking of grayscale and color images. *IEEE transactions on image processing*, 16(5):1395–1411, 2007.

[15] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[16] Hunter Gabbard, Chris Messenger, Ik Siong Heng, Francesco Tonolini, and Roderick Murray-Smith. Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy. *arXiv preprint arXiv:1909.06296*, 2019.

[17] Goëry Genty, Lauri Salmela, John M Dudley, Daniel Brunner, Alexey Kokhanovskiy, Sergei Kobtsev, and Sergei K Turitsyn. Machine learning and applications in ultrafast photonics. *Nature Photonics*, 15(2):91–101, 2021.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[19] Alexander Goldenshluger and Arkadi Nemirovski. On spatially adaptive estimation of nonparametric regression. *Mathematical methods of Statistics*, 6(2):135–170, 1997.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Stephen R Green, Christine Simpson, and Jonathan Gair. Gravitational-wave parameter estimation with autoregressive neural network flows. *Physical Review D*, 102(10): 104057, 2020.

[22] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379: 113741, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113741. URL `https://www.sciencedirect.com/science/article/pii/S0045782521000773`.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] John Henry Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[26] J Michael Hollas. *Modern spectroscopy*. John Wiley & Sons, 2004.

[27] Prashant V. Kamat. Absolute, arbitrary, relative, or normalized scale? how to get the scale right. *ACS Energy Letters*, 4(8):2005–2006, 2019. doi: 10.1021/acsenergylett.9b01571. URL `https://doi.org/10.1021/acsenergylett.9b01571`.

[28] Emmanuel Kerrinckx, Laurent Bigot, Marc Douay, and Yves Quiquempois. Photonic crystal fiber design by means of a genetic algorithm. *Optics Express*, 12(9):1990–1995, 2004.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Florian Knoll, Kerstin Hammernik, Chi Zhang, Steen Moeller, Thomas Pock, Daniel K Sodickson, and Mehmet Akcakaya. Deep-learning methods for parallel magnetic resonance imaging reconstruction: A survey of the current approaches, trends, and issues. *IEEE signal processing magazine*, 37(1):128–140, 2020.

[31] Yury Korolev and Jonas Latz. Inverse Problems, Lectures notes Michaelmas term 2020. *University of Cambridge*, 2020. URL `https://www.damtp.cam.ac.uk/research/cia/files/teaching/Inverse_Problems_2020/LectureNotes2020.pdf`. Accessed: 2021-09-27.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[33] Andrey Kurenkov. A brief history of neural nets and deep learning. *Skynet Today*, 2020.

[34] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[36] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis, University of Helsinki*, pages in Chapter 6–7 and pages 58–60, 1970.

[37] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[38] G Moltó, M Arevalillo-Herráez, C Milián, M Zacarés, V Hernández, and A Ferrando. Optimization of supercontinuum spectrum using genetic algorithms on service-oriented grids. In *Proceedings of the 3rd Iberian Grid Infrastructure Conference*, pages 137–147, 2009.

[39] RR Musin and AM Zheltikov. Designing dispersion-compensating photonic-crystal fibers using a genetic algorithm. *Optics Communications*, 281(4):567–572, 2008.

[40] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[41] Gregory Ongie, Ajil Jalal, Christopher A Metzler, Richard G Baraniuk, Alexandros G Dimakis, and Rebecca Willett. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020.

[42] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL `https://www.sciencedirect.com/science/article/pii/0041555364901375`.

[43] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[44] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[45] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993. doi: 10.1109/ICNN.1993.298623.

[46] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[47] Lauri Salmela, Nikolaos Tsipinakis, Alessandro Foi, Cyril Billet, John Dudley, and G. Genty. Predicting ultrafast nonlinear dynamics in fibre optics with a recurrent neural network. *Nature Machine Intelligence*, 3:1–11, 04 2021. doi: 10.1038/s42256-021-00297-z.

[48] Davide Salvi, Sebastian Gonzalez, Fabio Antonacci, and Augusto Sarti. Parametric optimization of violin top plates using machine learning. *arXiv preprint arXiv:2102.07133*, 2021.

[49] SchedMD. Slurm Workload Manager. `https://slurm.schedmd.com`, 2021. Accessed: 2021-09-27.

[50] Andrea Schillaci, Maurizio Quadrio, Carlotta Pipolo, Marcello Restelli, and Giacomo Boracchi. Inferring functional properties from fluid dynamics features. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4091–4098. IEEE, 2021.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Stanford Vision Lab. ImageNet. `https://image-net.org/`, 2020. Accessed: 2021-09-27.

[53] Diego Stucchi, Andrea Corsini, Goëry Genty, Giacomo Boracchi, and Alessandro Foi. A weighted loss function to predict control parameters for supercontinuum generation via neural networks. In *2021 IEEE 31th International Workshop on Machine Learning for Signal Processing Proceeding (MLSP)*. IEEE, 11 2021.

[54] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[55] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[56] Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. SIAM, 2005.

[57] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. Also online at: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Accessed: 2021-09-27.

[58] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[59] Nanzhe Wang, Haibin Chang, and Dongxiao Zhang. Deep-learning-based inverse modeling approaches: A subsurface flow example. *Journal of Geophysical Research: Solid Earth*, 126(2):e2020JB020549, 2021.

[60] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[61] Zhongwei Xu and Alessandro Foi. Anisotropic denoising of 3d point clouds by aggregation of multiple surface-adaptive estimates. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[62] Wen Qi Zhang, Shahraam Afshar, and Tanya M Monro. A genetic algorithm based approach to fiber design for high coherence and large bandwidth supercontinuum generation. *Optics Express*, 17(21):19311–19327, 2009.

[63] Wen Qi Zhang, Jay E Sharping, Richard T White, Tanya M Monro, and Shahraam Afshar. Design and optimization of fiber optical parametric oscillators for femtosecond pulse generation. *Optics Express*, 18(16):17294–17305, 2010.

List of Figures

List of Tables