eman ta zabal zazu

Universidad del País Vasco — Euskal Herriko Unibertsitatea

Ph.D. Dissertation *Doktorego tesia*

Solving Partial Differential Equations using Artificial Neural Networks

Deribatu partzialetako ekuazioen ebazpena neurona-sare artifizialen bidez

Carlos Uriarte

2023

Ph.D. Dissertation

Solving Partial Differential Equations using Artificial Neural Networks

Carlos Uriarte

Supervisors: David Pardo, Elisabete Alberdi

2023

Doktorego tesia

Deribatu partzialetako ekuazioen ebazpena neurona-sare artifizialen bidez

Carlos Uriarte

Zuzendariak: David Pardo, Elisabete Alberdi

2023

This dissertation is written in English and Basque. In the part prior to the main body, the mixture of languages obeys the following rule: for contents with a single paragraph, two paragraphs will appear—the first one in English and the second one in Basque. For contents with more than one paragraph, they will be distinguished by subsections entitled in either English or Basque. In the body of the dissertation, odd chapters are developed in Basque while even chapters are in English. To internationalize the scope of the presented work, the appendices include the English translations of the odd chapters written in Basque.

Tesi hau ingelesez eta euskaraz idatzita dago. Tesiaren gorputzaren aurreko zatian, hizkuntzen nahasketak honako arau hau beteko du: bi paragrafo agertuko dira (lehenengoa ingelesez eta bigarrena euskaraz) testu-paragrafo bakarra duten edukietan. Paragrafo bat baino gehiagoko edukiak aldiz, ingelesezko edo euskarazko izenburua duten azpiatalen bidez bereiziko dira. Tesiaren gorputzean, kapitulu bakoitiak euskaraz garatuko dira, eta kapitulu bikoitiak, ingelesez. Aurkeztutako lanaren irismena nazioartera zabaltzeko xedez, euskarazko kapitulu bakoitien ingelesezko itzulpenak eranskinetan jasoko dira.

Contents/Aurkibidea

Acknowledgments/Esker onak

Acknowledgements

Over the period of completion of this dissertation, I have met people to whom I will always be grateful for many and diverse reasons.

First of all, I want to express my deepest gratitude to David: a singular guy with an assertive and carefree attitude, and an unusual way of looking at life. These, combined with his patience, incredible interpersonal skills, and great intuition, make him a wonderful person to work with, be advised by, and enjoy funny and not-so-funny moments. I thank him for: his continuous teachings, our long conversations, and that innate discipline of him to destroy works that had cost me so much to build—to tell the truth, he used to be right. I will be eternally grateful to him for the successes and stumbles I have found along this exciting and unrepeatable PhDing journey.

I would also like to thank my other supervisor, Elisabete, for bringing me the opportunity to develop the thesis under her supervision. I would especially like to thank her for her impeccable follow-up during the organization and writing of this dissertation. Without her, it would have been an infinitely more complex task.

In addition to my supervisors, much of the credit for the sublime working environment goes to the rest of my research colleagues. In particular, Javier was a fundamental workmate during the first stage of my PhD, always available to help me virtually during the pandemic. His virtuosity in tackling tedious tasks without tiring fascinates me, as well as in continuously proposing to have “a last beer”—which rarely was the last one—after a day of hard work. I have learned enormously from him and shared unforgettable moments. The same goes for Ana, who accompanied me from the beginning to the end of my predoctoral career and with whom I have also shared enjoyable moments in every professional and not-so-professional appointment. To highlight some of the not-so-professional ones: our adventures during the “Camino de Santiago” and the “Aste Nagusiak” full of glitter.

I am incredibly grateful to Judit for hosting me in such an endearing way during my research stay at the Oden Institute for Computational Sciences and Engineering of the University of Texas at Austin, USA. It was an adventure I enjoyed a lot. I also thank her for introducing me to Prof. Demkowicz, from whom I learned a great deal of knowledge I have applied in my research works. Similarly, I thank Ignacio, Paulina, Keko, and Patrick for their incredible kindness and hospitality during my research stay at the Institute of Mathematics of the Pontifical Catholic University of Valparaíso, Chile. I have many good anecdotes engraved in my memory in which they were taking part, particularly at moments of “carreteo”.

It is difficult to sufficiently acknowledge all the lived experiences with so many colleagues during these four years. To avoid making this section too long, I briefly thank Jamie, Jon Ander, Julen, Lena, Oscar, Felipe, Mahdi, Ali, Jesús, Manuela, and Tomás. Although the greeting is brief, it has a lot of meaning. I thank them all a lot.

I would also like to thank all the BCAM staff for assisting me during the bureaucratic difficulties encountered, which have not been few. In this regard, I want to thank Miguel especially. He has been infinitely patient and lovely with me on many occasions. A thousand thanks!

Besides, during these four years, I have met people who, although not directly related to my research environment, have become friends of mine due to their relationships with workmates and significantly contributed to the easygoingness of this adventure. I particularly want to thank Claudio and Mario for our many good times together.

Finally, I want to thank my family and friends for their support. In particular, I want to offer my eternal gratitude to my father, Enrique; my brother, Luis; and my partner, Iratxe, who unconditionally accompanied me and firsthand experienced my moments of tension and joy. Thank you for understanding the conflict of a researcher lifestyle—working at ungodly hours, leaving aside the usual healthy social habits, simply because I felt I could not procrastinate my moments of research inspiration. Heartfeltly, thanks.

Esker Onak

Doktoretza egiten hasi nintzenetik, arrastoa utzi didaten hainbat pertsona ezagutu ditut bidean, eta guztiei adierazi nahiko nieke nire esker ona.

Lehenik eta behin, nire esker onik handiena adierazi nahi diot Davidi. David pertsona berezia da, jarrera asertiboa eta axolagabea dauka, eta bizitza ezohiko eran ikusten du. Ezaugarri horiek, haren pazientziarekin, haren trebetasun interpertsonal harrigarriekin, eta bere-berea duen intuizio handiarekin batera, pertsona benetan zoragarria egiten dute berarekin lan egiteko orduan, harengandik aholkuak jasotzeko orduan, eta une dibertigarrietan eta ez hain dibertigarrietan gozatzea posible izaten da. Eskerrak eman nahi dizkiot, hain zuzen ere, haren etengabeko irakaspenengatik, gure elkarrizketa luzeengatik, eta egitea hainbeste kostatu zitzaizkidan lanak desegiteko duen berezko diziplinagatik (egia esan, arrazoia izaten zuen gehienetan). Beti egongo naiz berari eskertuta doktorego-bidaia liluragarri eta errepikaezin honetan aurkitu ditudan arrakasta eta behaztopengatik.

Aldi berean, eskerrak eman nahi dizkiot nire beste zuzendariari, Elisabeteri, bere ardurapean tesia garatzeko aukera emateagatik. Batez ere, eskerrak eman nahi dizkiot tesi honen antolaketan eta idazketan egin duen jarraipenagatik. Bera gabe, askoz lan konplexuagoa izango zatekeen.

Nire zuzendariei ezezik, laneko giro bikaina lankideei ere zor diet. Zehazki, Javier funtsezko lankidea izan zen nire doktoretzaren lehenengo fasean; beti egoten zen prest pandemian zehar era birtualean laguntzeko. Lan aspergarri eta nekagarrietan duen diziplinak liluratu egiten nau, eta ez ditut ahaztu nahi gogor lan egindako egunen amaieran “azken garagardoa” hartzeko proposamenak (gutxitan azkena izaten zena). Asko ikasi dut harengandik eta une ahaztezinak partekatu ditugu. Anari ere eskerrak eman nahi dizkiot, nire karrera aurredoktoralean hasieratik bukaeraraino ondoan egon den lankidea izan baita, eta une gozagarriak partekatu baititut hitzordu profesional eta ez hain profesional guztietan. Une ez oso profesionalen artean, Donejakue bideko abenturak eta Aste Nagusietako distiraz betetako denboraldiak dira aipagarrienetakoak.

Asko eskertzen diot Juditi hainbeste une berezi partekatzegatik nire egonaldian zehar Oden Konputazioko eta Ingeniaritzako Institutuan, Texasko Unibertsitatean Austinen, Estatu Batuetan. Asko gustatu zitzaidan abentura izan zen. Halaber, eskerrak eman nahi dizkiot Demkowicz irakaslea aurkezteagatik, nire ikerlanetako askotan aplikatu dudan ezagutzaren garatzailea hura baita. Era berean, eskerrak eman nahi dizkiet Ignacio, Paulina, Keko eta Patrick ikerlariei, Matematikako Institutuan Valparaísoko Pontifizia Katolika Unibertsitatean nire egonaldian izan zuten adeitasun eta abegikortasunagatik. Anekdota on asko grabatuta dauzkat oroimenean, batez ere “carreteo” garaietakoak.

Zaila da lau urte hauetan hainbeste lankiderekin bizitako esperientzia guztiak behar bezala aitortzea. Atal hau gehiegi ez luzatzearren, labur bada ere, eskerrak eman nahi dizkiet Jamie, Jon Ander, Julen, Lena, Oscar, Felipe, Mahdi, Ali, Jesús, Manuela eta Tomás lankideei. Nahiz eta agurra laburra izan, esanahi handikoa da.

Era berean, eskerrak eman nahi dizkiet BCAMeko administrazioko langile guztiei, hain gutxi izan ez diren zailtasun burokratikoen aurrean laguntza emateagatik. Bereziki, Migueli. Oso maitagarria eta pazientzia handikoa izan da nirekin askotan. Mila esker!

Gainera, lau urte hauetan, nire ikerketako ingurunearekin lotura zuzenik izan ez arren, egindako adiskideei ere eskerrak eman beharrean nago. Bereziki, Claudiori eta Mariori elkarrekin izan ditugun momentu itzelengatik.

Azkenik, eskerrak eman nahi dizkiet nire etxekoei eta adiskideei emandako laguntzagatik. Bereziki, esker onak nire aita Enriqueri, Luis anaiari eta nire bikotekide Iratxeri, baldintzarik gabe lagundu didatelako eta nire tentsio- eta poztasun-uneak bertatik bertara bizi izan dituztelako. Hain zuzen, eskerrik asko ikertzaile baten bizimoduak dakarren gatazka ulertzeagatik; hau da, ezorduetan lan eginez, ohiko gizarte-ohitura osasungarriak alde batera utzita, ikerketa-inspirazioko uneak atzeratzerik sentitzen ez nuelako. Bihotz-bihotzez, eskerrik asko.

Abstract/Laburpena

Abstract

Partial differential equations have a wide range of applications in modeling multiple physical, biological, or social phenomena. Therefore, we need to approximate the solutions of these equations in computationally feasible terms. Nowadays, among the most popular numerical methods for solving partial differential equations in engineering, we encounter the finite difference and finite element methods. An alternative numerical method that has recently gained popularity for numerically solving partial differential equations is the use of artificial neural networks.

Artificial neural networks, or neural networks for short, are mathematical structures with universal approximation properties. In addition, thanks to the extraordinary computational development of the last decade, neural networks have become accessible and powerful numerical methods for engineers and researchers. For example, imaging and language processing are applications of neural networks today that show sublime performance inconceivable years ago.

This dissertation contributes to the numerical solution of partial differential equations using neural networks with the following two-fold objective: investigate the behavior of neural networks as approximators of solutions of partial differential equations and propose neural-network-based methods for frameworks that are hardly addressable via traditional numerical methods.

As novel neural-network-based proposals, we first present a method inspired by the finite element method when applying mesh refinements to solve parametric problems. Secondly, we propose a general residual minimization scheme based on a generalized version of the Ritz method. Finally, we develop a memory-based strategy to overcome a usual numerical integration limitation when using neural networks to solve partial differential equations.

Laburpena

Deribatu partzialetako ekuazioak aplikazio ugari dituzte fenomeno fisiko, biologiko edo sozial anitzen modelizazioan. Horregatik, funtsezkoa da ekuazio horien soluzioen hurbilpenak konputazionalki egingarriak diren terminoetan adieraztea. Gaur egun, ingeniaritzan, deribatu partzialetako ekuazioak ebazteko zenbakizko metodo ezagunenen artean daude diferentzia finituen eta elementu finituen metodoak. Duela gutxi, deribatu partzialetako ekuazioak ebazteko ospea hartu duen zenbakizko metodo bat neurona-sare artifizialen erabilpena da.

Neurona-sare artifizialak edo, laburtzearren, neurona-sareak hurbilketa-propietate unibertsalak dituzten egitura matematikoak dira. Gainera, azken hamarkadako garapen konputazional apartari esker, ingeniarientzako eta ikertzaileentzako zenbakizko metodo eskuragarri eta indartsu bihurtu dira. Adibidez, irudien eta hizkuntzen prozesamendua neurona-sare artifizialen gaur egungo aplikazioak dira, eta duela urte batzuk pentsaezina zen errendimendu bikaina erakusten dute.

Tesi honetan neurona-sare artifizialen bidezko deribatu partzialetako ekuazioen zenbakizko ebazpena aztertuko dugu, honako helburu bikoitzarekin: alde batetik, neurona-sareen portaera ikertzea deribatu partzialetako ekuazioen soluzioen hurbiltzaile gisa, eta bestetik, neurona-sareetan oinarritutako metodoak proposatzea, ohiko zenbakizko metodoen bidez nekez ekin dakiekeen lan-esparruetarako.

Neurona-sareetan oinarritutako proposamen berritzaile gisa, lehenik eta behin, elementu finituen metodoaren funtzionamenduan oinarritutako metodo bat aurkeztuko dugu, problema parametrikoak ebazteko diskretizazioan fintzeak aplikatzen direnean. Bigarrenik, hondarren minimizazio eskema orokor bat proposatuko dugu, Ritz-en metodoaren bertsio hedatu batean oinarritua. Azkenik, memorian oinarritutako zenbakizko integrazio teknika bat erakutsiko dugu. Teknika horren helburua da deribatu partzialetako ekuazioak ebazteko neurona-sareak erabiltzen direnean agertzen den ohiko muga gainditzea.

Acronyms/Akronimoak

Throughout this dissertation, English acronyms will be used exclusively, regardless of whether they appear in English or Basque portions of text. In English, the first letter of each constituent word is capitalized. In Basque, only proper or foreign constituent words have the first letter capitalized (or at the beginning of a sentence). In the following list, acronyms appear in singular form in English, while in bare form and between parentheses in Basque.*

Tesi honetan zehar, ingelesezko akronimoak bakarrik erabiliko dira, horiek agertzen diren testu zatia ingelesez ala euskaraz idatzita dagoen kontuan hartu gabe. Ingelesez, lehen letra larriaren irizpideari eutsiko diogu hitz eratzaile guztietan. Euskaraz, hitz eratzaileen lehen letrak larriz egongo dira izen berezien kasuan bakarrik (edo esaldiaren hasiera bada). Hurrengo zerrendan, akronimoak singularrean agertzen dira ingelesez, eta euskaraz, berriz, oinarriko forman eta parentesi artean.*

1D one-dimensional (dimentsio bakar)

2D two-dimensional (bi dimentsio)

3D three-dimensional (hiru dimentsio)

AD Automatic Differentiation (Deribaketa automatiko)

ANN Artificial Neural Network (Neurona-sare artifizial)

BVP Boundary-Value Problem (Mutur-balioen problema)

D^{2}RM Deep Double Ritz Method (Deep Ritz metodo bikoitz)

DeepFEM Deep Finite Element Method (Deep elementu finituen metodo)

DNN Deep Neural Network (Neurona-sare sakon)

DRM Deep Ritz Method (Deep Ritz metodo)

FDM Finite Difference Method (Diferentzia finituen metodo)

*With some exceptions/Salbuespenak salbuespen.

FEM Finite Element Method (Elementu finituen metodo)

FFNN Feed-Forward Neural Network (Aurrerantz elikatutako neurona-sare)

GDRM Generalized Deep Ritz Method (Deep Ritz metodo orokortua)

GPU Graphics Processing Unit (Prozesamendu grafikoko unitate)

MC Monte Carlo

NN Neural Network (Neurona-sare)

ODE Ordinary Differential Equation (Ekuazio diferentzial arrunt)

PDE Partial Differential Equation (Deribatu partzialetako ekuazio)

PINN Physics-Informed Neural Network (Fisikatik informatutako neurona-sare)

SGA Stochastic Gradient Ascent (Gradiente igoera estokastiko)

SGD Stochastic Gradient Descent (Gradiente jaitsiera estokastiko)

SGDM SGD with Momentum (SGD momentum-arekin)

TF TensorFlow

TF2 TensorFlow 2

UAT Universal Approximation Theorem (Hurbiltasun unibertsalaren teorema)

WANs Weak Adversarial Networks (Sare aurkari ahul [pluralean])

List of Figures/Irudien zerrenda

List of Tables/Taulen zerrenda

Chapters/Kapituluak

1. Sarrera[*]

1.1. Motibazioa

Deribatu partzialetako ekuazioek (PDE-ek)[1] balio handia dute gizartearentzat fenomeno biologiko, fisiko edo sozial anitzak modelatzeko duten gaitasun zabalagatik [73, 207, 70, 192]. Ekuazio horietan deribatuak erabiltzeak aukera ematen digu erlazio konplexuak eta haien denboraren eta/edo espazioaren gaineko aldaketaren tasak deskribatzeko. Adibidez, honako fenomeno hauek PDE-en bidez modelizatutako ereduak onartzen dituzte: bero-transferentzia [102, 88], eremu elektromagnetikoak [106, 221], fluidoen dinamika [23, 14, 223], populazioen eboluzioa [96, 160, 38], eta finantza [26, 101] edo osasun [193, 141, 165] arloko iragarpenak.

Hala ere, egoera fisiko edo sozial bat deskribatzeko ekuazio bat ezartzea lehen urratsa baino ez da. Arazo mota ezberdinak sortzen dira PDEak erabiltzean, eta horietara hurbiltzeko modua eskura dauden konputazio-baliabideen araberakoa da.

Aurreranzko problemetan, PDE bat eta hasierako edo mugaldeko baldintza batzuk betetzen dituen funtzioa zehaztea da interesatzen zaiguna. Horrela, aurreranzko problemen soluzioek modelatutako fenomenoaren buruzko ezagutza eman diezagukete, eta hori erabil daiteke iragarpenak egiteko, normalean soluziotik hartutako post-prozesamenduetako neurketen bidez.

Alderantzizko problemetan, soluzioaren neurriak ditugu, baina ekuazioan parametro espezifikoek duten balioa aurkitu nahi izaten da. Zeharkako arazo bat da, emandako neurketekin bat datorren ebazpen bat ematen duten parametroak zehaztu behar baitira [210]. 1.1. adibidean eremu elektromagnetikoetan dauden aurreranzko eta alderantzizko problemen eredu posible bat erakusten da [72].

[*]See Appendix A for the English version/Ikus A. eranskina ingelesezko bertsiorako.

[1]Ingelesetik, *Partial Differential Equation (PDE)*.

1.1. adibidea (Eremu elektromagnetikoak)**.** Maxwellen ekuazioek lau lege fisiko ezagunen arabera modelatzen dute eremu elektromagnetikoa [72, 106]:

$$\begin{cases} \nabla \times \mathbf{E} = -j\omega\boldsymbol{\mu}\,\mathbf{H} - \mathbf{M}, & \text{Faradayren Legea,} \\ \nabla \times \mathbf{H} = (\boldsymbol{\sigma} + j\omega\boldsymbol{\varepsilon})\mathbf{E} + \mathbf{J}, & \text{Amperèren Legea,} \\ \nabla \cdot (\boldsymbol{\varepsilon}\mathbf{E}) = \rho_f, & \text{Gaussen Elektrizitatearen Legea,} \\ \nabla \cdot (\boldsymbol{\mu}\mathbf{H}) = 0, & \text{Gaussen Magnetismoaren Legea.} \end{cases} \tag{1.1}$$

Hemen, **E** eta **H** elektrizitatearen eta magnetismoaren eremuak dira frekuentzien domeinuan, hurrenez hurren; **J** eta **M** iturri terminoak dira; $\boldsymbol{\sigma}$ ingurunearen eroankortasun elektrikoa da; $\boldsymbol{\varepsilon}$ permitibitate elektrikoa da; $\boldsymbol{\mu}$ iragazkortasun magnetikoa da; ρ_f karga elektrikoaren dentsitatea da; j unitate irudikaria da; eta ω frekuentzia angularra da. Orduan, eremu elektromagnetikoen aurreranzko eta alderantzizko eredu posible bat $\{\boldsymbol{\sigma}, \boldsymbol{\mu}, \boldsymbol{\varepsilon}\}$ parametroen eta $\{\mathbf{Z}\}$ neurketen mende osa daiteke, **Z** inpedantzia tentsorea izanik eta honako era honetan definituta egonik: $\mathbf{E} = \mathbf{Z}\mathbf{H}$. Xehetasun gehiagoren bila, ikus, adibidez, [9]. 1.1. irudian eredu honen laburpen grafikoa erakutsi da.

$$\begin{tikzcd} \begin{array}{c}\text{Parametroak}\\ \{\boldsymbol{\sigma}, \boldsymbol{\mu}, \boldsymbol{\varepsilon}\}\end{array} \arrow[r, "\text{Ebatzi PDEa}"] \arrow[rr, bend left, "\text{Aurreranzko problema}"] & \begin{array}{c}\text{Soluzioa}\\ \{\mathbf{E}, \mathbf{H}\}\end{array} \arrow[r, "\text{Post-prozesaketa}"] & \begin{array}{c}\text{Neurketak}\\ \{\mathbf{Z}\}\end{array} \arrow[ll, bend left, "\text{Alderantzizko problema}"] \end{tikzcd}$$

1.1. irudia: Aurreranzko eta alderantzizko problemen diagrama eremu elektromagnetiko batean.

Aurreranzko problemak ebazteko, orokorrean, alderantzizko problemak ebazteko baino baliabide gutxiago behar dira. Behin parametroak aukeratuta, aurreranzko problema bat PDE simulazio bakar bat burutzean datza. Aitzitik, alderantzizko problema batek, aurreranzko-problemen urratsez urratseko ebazpena eskatzen du, urrats bakoitzean parametroen aukeraketa doituz aurreko aurreranzko simulazioan lortu den emaitzaren arabera [210, 103].

Bi arazo motak lotuz, problema *parametrikoak* ditugu, non helburua soluzio edo neurketen portaera parametroen funtzio gisa aztertzea baita. Soluzioa parametro baten edo batzuen araberakoa izan daiteke, eta parametro horien aldaketek sistemaren portaerari nola eragiten dioten azter dezakegu. Alternatiboki, ikuspegi parametrikoa aurreranzko problema gisa har daiteke, non ekuazioaren parametroek soluzioaren aldagaien rola hartzen baitute. Tipikoki, aurreranzko problema parametrikoa ez da lineala sartutako aldagai berriekiko; eta horrek problema ebazteko gaitasuna izugarri mailakatzen eta eragozten du.

Tesi honetan, PDE lineal parametrikoak eta ez parametrikoak ebazteko zereginari soilik helduko diogu, non aplikazio eta lan-esparru askotan kritikoa baita. Gainera, PDEak esango diegu bai dimentsio bakarreko ekuazio diferentzialei —oro har, *ekuazio diferentzial arruntak (ODEak)*[2] deituak— bai eta dimentsio handiagokoei ere.

1.2. Zenbakizko metodo tradizionalak

PDEak ebazteko hainbat metodo daude, oro har, bi kategoriatan sailka daitezkeenak: metodo analitikoak eta zenbakizkoak.

Metodo analitikoek PDEaren soluzio zehatz bat aurkitzeko teknika matematikoak erabiltzea eskatzen dute, hala nola, aldagaien bereizketa, transformatu integralak edo analisi konplexua [205, 87, 213]. Hala ere, metodo horiek, askotan, PDE eta geometria nahiko sinpleetara mugatzen dira. Gainera, PDEa analitikoki ebatz daitekeenean ere, gerta daiteke emaitzaren adierazpena konplikatua izatea (adibidez, potentzia infinitu sorta baten moduan adieraztea). Horrelakoetan, zaila izan daiteke hori interpretatzea eta kasu praktikoetan aplikatzea.

Bestalde, zenbakizko metodoak PDEaren soluzioa hurbiltzen dute algoritmo konputazionalen bidez. Oro har, metodo hauek erraz interpreta edo berreskura daitezkeen hurbilketak proposatzen dituzte (adibidez, funtzio sinpleen konbinazio lineal finituak erabilita). Jarraian, labur aztertuko ditugu PDEak ebazteko erabiltzen diren hiru zenbakizko metodo ezagunen alderdi nagusiak.

Diferentzia finituen metodoak (FDMak)[3] PDEaren domeinu espaziala eta/edo denbora tartea azpidomeinuen kopuru finitu batean banatzen du [203, 178, 130]. Orduan, puntu diskretu horietako deribatuen ebaluazioak diferentzia finituak eta hurbileko puntuen balioak dituzten ekuazio aljebraikoak ebatziz hurbiltzen dira. FDMak PDE bat ekuazio linealen sistema batean bihurtzen du, aljebra matrizialeko tekniken bidez ebatz daitekeena. Kontzeptualki sinplea izan arren, desafiatzailea gerta daiteke arazo konplexuetarako edo domeinu irregularretarako diseinatzen denean. Zehazki, *dimentsioen maldizio* deritzona jasaten du [29, 30], non inplikatutako matrizearen tamaina esponentzialki hazten baita problemaren dimentsioarekin.

[2]Ingelesetik, *Ordinary Differential Equation (ODE).*
[3]Ingelesetik, *Finite Difference Method (FDM).*

Era berean, *elementu finituen metodoak (FEMak)*[4] [100, 39, 138, 184] diskretizazio-sareetan oinarritutako eskema bat proposatzen du, non soluzio hurbildua aurrez ezarritako funtzio batzuen konbinazio lineal finitu baten forman adierazita baitago (normalean, zatikako polinomioak) euskarri lokalekin. Euskarria berariaz diseinatzen da domeinuko eskualde espezifikoetan, elementu deiturikoetan, emaitza bezala lortzen den matrizea sakabanatua izateko. Bereziki, FEMa egokia da PDEak formulazio bariazionaletan daudenean, baina FDMaren antzeko zailtasunak ditu.

Metodo espektralak [81, 36, 46] zenbakizko metodo mota bat dira, eta euskarri globala duten oinarriko funtzio ortogonalen konbinazio linealen bidez hurbiltzen dituzte funtzioak, hala nola, Chebyshev polinomioak edo Fourier serieak. Metodo horiek oso zehaztasun handia lor dezakete eta bereziki erabilgarriak dira portaera oszilakor handiko soluzio leunetarako, baina konputazio-ikuspegitik bideraezinak izan daitezke geometria konplexuetarako.

Funtsean, aipatutako zenbakizko metodoek oinarri finitu-dimentsional bat aurrez ezartzea eta dagokion konbinazio linealaren koefizienteen bidez hurbilpena parametrizatzea dute lan-eremu. Diskretizazio-sare bat erabiltzen denean (FDMetan eta FEMetan), horrek behar bezain fina izan behar du soluzio zehatzaren portaera konplexuak behar bezala antzemateko, baina gehiegizko finketaren eragozpen konputazionalean erori gabe. Metodo espektraletan, erronkarik handiena da oinarri ortogonaleko funtzioak domeinu arbitrarioetarako hautatzea.

1.3. Neurona-sareak hurbiltzaile unibertsal gisa

Neurona-sare artifizialek (ANNek)[5], edo, laburtzeko, *neurona-sareek (NNek)*[6], funtzioak hurbiltzeko alternatiba bat eskaintzen dute, aurretik azaldutako ohiko konbinazio linealen ikuspegiarekiko ezberdina dena.

NNen formarik oinarrizkoena *aurrerantz elikatutako neurona-sareak (FFNNak)*[7] dira [190, 105, 12, 77, 195], garuneko NN biologikoen egituran eta funtzionamenduan inspiratzen direnak, eta elkarren artean konektatutako *neuronen* bidez osaturik daudenak. Neuronak geruzetan antolatzen dira, eta informazioa norabide batetik bestera doa, geruza batetik hurrengora. Neurona bakoitzak aurreko geruzako neuronen informazioa jasotzen du eta batura haztatu bat kalkulatzen du. Batura hori aurreko geruzako neurona *guztietatik* abiatuta kalkulatzen denean, FFNNa *guztiz konektatuta* dagoela esango dugu. Batuketa hori aktibazio ez-linealeko funtzio batetik pasarazten da, gero emaitza hurrengo geruzako neu-

[4]Ingelesetik, *Finite Element Method (FEM)*.
[5]Ingelesetik, *Artificial Neural Network (ANN)*.
[6]Ingelesetik, *Neural Network (NN)*.
[7]Ingelesetik, *Feed-Forward Neural Network (FFNN)*.

ronetara transmititzeko. Guztiz konektatutako FFNNen deskribapen formala 2. kapitulura atzeratuko dugu.

Guztiz konektatutako FFNNak hurbiltzaile unibertsalak dira. Era arinean esanda, emandako ia edozein funtzio nahi den adina hurbiltzeko gaitasuna dute.

Hurbiltasun unibertsalaren teoremaren (UATren)[8] lehen bertsioetako bat *zabalera arbitrarioko* kasua da. Bertan frogatzen da ezkutuko geruza bakarreko FFNNak funtzio jarraituen hurbiltzaile unibertsalak direla, behar bezain zabalak baldin badira. Emaitza hori paraleloki eta independenteki frogatu zen 1989an, "sigmoid" aktibazio funtzioetarako [54] eta aktibazio-funtzio ez-konstante, mugatu eta monotonoki gorakorretarako [98]. Azken lan horren egile nagusiak bi urte geroago frogatu zuen ez dela aktibazio-funtzioaren hautaketa espezifikoa FFNNei hurbiltzaile unibertsal izateko ahalmena ematen diena, baizik eta FFNNaren arkitektura bera [97]. Horrekin lotuta, geroago erakutsi zen baliokideak direla hurbilketa unibertsalaren ezaugarria eta aktibazio ez-polinomikoaren funtzioa izatea [129, 175].

UATrako zabalera arbitrarioaren kasuaren bertsio "duala" zabalera mugatua finkatzean eta *sakontasun arbitrarioa* ezartzean datza. Zenbait egilek formulazio hori aztertu dute 2000ko hamarkadaren hasieratik [83, 118], ReLU aktibazio-funtzioetan arreta berezia jarriz [230, 140, 90, 89]. Era berean, *zabalera eta sakonera mugatuko* emaitzak aurkitzen ditugu. Adibidez, [146]-k baieztatzen du FFNN baterako existitzen dela bi sakonerako eta zabalera mugatuko hurbiltasun unibertsalaren ezaugarriak dituen sigmoidal moduko aktibazio funtzio bat. Gainera, [85]-ek frogatu zuen geruza ezkutu bakarreko eta zabalera mugatuko FFNNek aldagai-bakarreko funtzioetarako hurbiltasun unibertsalak izaten jarraitzen dutela, oro har, aldagai-anitzeko funtzioetarako ez dena betetzen.

Azken hiru hamarkadetan, UATren hainbat hedapen garatu dira non ez-jarraituak diren aktibazio funtzioak [129], domeinu ez-trinkoak [118], eta arkitektura eta topologia alternatiboak landu baitituzte [118, 136]. Zehazki, [22]-k UATren bertsio kuantitatibo bat ezartzen du FFNNen *dimentsionaltasunaren maldizioa* gainditzeko gaitasuna aztertuz, zatika linealak diren funtzioen espazioekin alderatuta. Gai horri buruzko eztabaida zabalagoak aztertzeko, [176, 19, 24, 226] lanak eta horietan jasotako erreferentziak aipatzen ditugu.

Azken urteotan, hainbat ekimen eta ikerketek ebidentzia edo froga enpirikoen bidez erakutsi dute arkitektura mota alternatiboek edo haien arteko konbinazio egokiek FFNNen hurbiltzeko eta kalkulatzeko gaitasuna gaindi dezaketela. Besteak beste, arkitektura konboluzionalak, errepikariak eta hondarrarenak (ingelesezko, *convolutional*, *recurrent* eta *residual* terminoetatik eta CNN, RNN eta ResNet laburduren bidez adierazitako NNak, hurrenez hurren) dira nabarmenenak [52, 59, 132, 206, 77, 93, 222, 4, 67, 220].

[8]Ingelesetik, *Universal Approximation Theorem (UAT)*.

Tesian zehar, guztiz konektatutako FFNN arkitekturetara mugatuko gara. Beraz, NN eta FFNN terminoak bereizi gabe irakur daitezke hemendik aurrera, kontrakorik esaten ez bada. Gainera, nabarmentzekoa da erabilitako arkitekturaren hurbilpen gaitasunen azterketa tesi honen irismenetik kanpo geratzen dela. Bestela esanda, (era inozoan) onartuko dugu esperimentazioan zehar hautatutako arkitekturek "behar adina gaitasun dutela hurbiltzeko".

1.4. Berrikuspen literarioa

Azken urteotan, PDEak NNen bidez ebazteko lan ugari sortu dira.

1.4.1. NNak erabiliz PDEak ebazteko lehenengo lanak

1.3. ataleko terminoei jarraituz NNen bidez PDEak ebazteko lehenengo lan dokumentatua 1994koa[9] da [63]. Aurreko lan batzuek PDEak ebazteko NNak erabili arren (ikus, adibidez, [127, 216, 152]), [63] proposamena izan zen PDE baten soluzio baten irudikapena ustiatu zuen lehenengoa, hurbilketa unibertsalaren propietatean oinarritutako NN baten bidez.

Lan honek FFNN arkitektura proposatzen du, aktibazio sigmoidaleko funtzioekin, mutur-baldintzak dituen PDE baten soluzioa hurbiltzeko —normalean, *mutur-balioen problema (BVPa)*[10] deitzen dena—. Espezifikoki, BVPa honela deskribatuta dago:

$$\begin{cases} Au = f, & \Omega\text{-n}, \\ Du = g, & \partial\Omega\text{-n}, \end{cases} \tag{1.2}$$

non Ω domeinu ireki eta mugatua baita, $\partial\Omega$ domeinuaren muga da, f emandako iturri-funtzio bat da, eta A eta D operadore (diferentzial/mugadun) jakin batzuk dira. Egileek honako hau adierazi zuten hitzez hitz (itzulita)[11]: "(1.2). ekuazioaren soluzio sendo bat $\mathbb{U}$ espazioko funtzioen elementu bat da, PDEa eta mutur-baldintzak betetzen dituena. [...] Hala ere, $\mathbb{U}$ espazio horretan hurbiltzaile 'unibertsal' bat baldin badago (hau da, behar adina parametro dituen eta espazioko edozein elementurekiko behar adina hurbil dagoen funtzio bat badago), orduan, hurbiltzaile hori bilatzen ari garen soluziorako aukera ona izan daiteke. Bi funtzioren arteko hurbiltasuna $\mathbb{U}$-ko dentsitatearen kontzeptuaren bidez azaltzen da. Definizio matematiko zehatz baterako, ikus Hornik et al. [98]".

[9]Argitaletxeak 1992ko abuztuan jaso zuen, 1993ko abuztuan berrikusi zen eta 1994ko martxoan argitaratu zen lehen bertsioa.

[10]Ingelesetik, *Boundary-Value Problem (BVP)*.

[11]Jatorrizko sinboloak eta erreferentziak egokitu egin dira gure aurkezpenerako. [...] ikurrak esan nahi du jatorrizko testuaren zati bat ez dela jaso.

Orduan, PDEaren ebazpen-zeregina hurrengo *funtzio objektiboaren* minimizazio bezala ezar daiteke:

$$\mathcal{F}(u_{\mathrm{NN}}) = \int_{\Omega} \|Au_{\mathrm{NN}} - f\|^2 + \int_{\partial\Omega} \|Du_{\mathrm{NN}} - g\|^2, \tag{1.3}$$

non u_{NN} FFNN baita. Jatorriz, $\|\cdot\|$ zehaztugabea zen, baina, ziurrenik, 2-norma diskretutzat hartuko zen $\mathcal{F}$-ren minimizazioa L^2-minimizazio-eskeman uler dadin ([53]-n iradokitzen den moduan).

Aurkezpen horren ondoren, egileek esan zuten[12]: "Orain, Au eta Du modu itxian kalkula daitezke x terminoaren mende. Ω-n eta $\partial\Omega$-n puntuak hautatzen ditugu eta (1.3) formula aplikatzea proposatzen dugu. Beraz, funtzio objektiboa minimiza dezakegu [NNtik ikas daitezkeen parametroetarako balioak aurkitzeko (hortik dator (1.2)-ren soluzio hurbildua)]". Egileek zehaztu zuten funtzio objektiboaren minimizazioa "ia-Newton metodo" baten arabera eta "diferentzia finituko gradienteak"erabiliz egin zutela (ikus, adibidez, [61] eta [156] erlazionatutako erreferentzia gisa).

Harrezkero, hainbat lan sortu ziren NNak erabiliz PDE ebazpenaren esparruan. Jarraian, 1990eko hamarkadaren amaieratik 2010eko hasierara arteko garrantzitsuenetako batzuk errepasatuko ditugu.

1994an, [154, 153, 71] lanek FFNN bat erabiltzea proposatu zuten, geruza ezkutu bakarreko arkitekturarekin eta "muga gogorreko"aktibazio funtzioak erabiliz (jatorriz horrela deitu zieten egileek), PDE linealak eta ez-linealak ebazteko. Aktibazio-funtzio hori hautatzeak aukera ematen du NN guztiak zatika-linealak diren funtzio familia gisa ulertzeko, eta, horrela, PDEaren soluzioa termino horietan hurbiltzeko. 1998an, [124] eskema bat proposatu zuen mutur-baldintzak modu "gogor" batean inposatzeko, FFNN (muturrik gabeko) arkitektura filtro batekin konposatzean ([63]-n ez bezala, non mutur-baldintzak "leunki" zehazten baitziren helburu-funtzioaren bitartez). 1999an, [164]-k hainbat emaitza eta diseinu dokumentatu zituen NN bidezko inplementazio ezberdinak erabilita, aldagai anitzeko funtzioen eta horien deribatu partzialen zenbakizko hurbilketarako. Horien artean, PDE aplikazioak daude, fluxu biskoelastikoak aztertzeko inguraketa-elementuen metodo baten parte gisa. 2000. urtean, NNetan oinarritutako eredu bat aurkeztu zen lehen ordenako zenbait PDE ebazteko, kontrol-sistema ez-linealetan interes berezia zuena, eta [125]-ek FFNN motako bi arkitektura ezberdinez osatutako eredua aurkeztu zuen, geometria irregularretan mutur-baldintzak dituzten PDEak ebaztera iristeko.

2001etik 2010eko hamarkadaren erdialdera, arestian aipatutakoen jarraipentzat har daitezkeen hainbat lan daude. Zehazki, [124]-k eta [94]-k aurreka-

[12]Lehen bezala, jatorrizko sinboloak eta erreferentziak egokituta daude. Kortxete arteko testuzatiak, [`testua`], jatorrizko testua aldatu edo ordezkatu dugula adierazten du ulergarriagoa izateko asmoarekin.

ri esanguratsuak ezarri zituzten arkitektura-diseinuetarako eta entrenamendu-konfigurazioetarako, hurrenez hurren, honako lan hauetarako: [1]-ek PDEak ebazteko NNak eta algoritmo ebolutiboak eta horien mutur-baldintzak konbinatzen dituen metodo bat proposatu zuen; [202]-k NNak erabili zituen Kuramato-Sivashinsky eta Navier-Stokes ekuazio ez-linealen dinamika aztertzeko; [147]-k metodo hibrido bat aurkeztu zuen ordena handiko PDEak ebazteko; [199]-k Schrödingerren ekuazioa landu zuen FFNN erabilita, energia-funtzioaren eskema hobetu batekin eta gainbegiratu gabeko entrenamenduaren bidez; [28]-k minimizazio-tekniken eta kokapen-metodoen konbinazio bat proposatu zuen NNen bidez soluzioetarako hurbilpen itxiak ezartzeko; [215]-ek FFNNak erabili zituen bilakaera gramatikalarekin eta optimizazio lokalarekin batera, ODE eta PDE sistemak ebazteko (ikus [214] bilakaera gramatikalean oinarritutako NNan entrenamenduan izandako aurrekari baterako); eta [151]-k NNan oinarritutako metodo bat aurkeztu zuen, domeinu irregularrak dituzten PDEak ebazteko, mutur-baldintza mistoekin. Gainera, PDE-en bereizmen-esparruen bi luzapen mota nagusi ditugu NNetan oinarrituak: oinarri-erradiala duten NNak[13] [142, 143, 108, 144, 109, 113, 145, 76, 13, 75, 47, 121]; eta FEMean inspiratutako NNak [31, 60, 234, 183, 148, 15, 110, 120] —batzuetan *elementu finituen neurona-sareak* deituak—. Aipatutako lan asko berrikusteko, [227] kontsulta daiteke.

1.4.2. NNen bidez PDEak ebazten TensorFlow-tik aurrera

TensorFlow (TF) [3, 2] kode irekiaren lehen bertsioaren aurretik 2015ean, *diferentziaketa automatiko (ADa)*[14] eraginkorrik ez egoteak erronkak planteatzen zituen NNetan oinarritutako ikerketan. Gaur egun, ADa funtsezkoa da NNen gradienteak kalkulatzeko entrenamenduan zehar [82, 27, 150]. Hori ez zegoenean, ikertzaileek eta profesionalek eskuz diseinatu behar izaten zuten gradienteen kalkulu-eskema, eta horrek, normalean, denbora asko eramaten zuen eta akatsak egiteko joera areagotu egiten zen, batez ere ereduak konplexuak ziren heinean. Ondorioz, berrikuntza eta esperimentazioa oztopatu egiten ziren. Ideia berriak eta eredu mota desberdinak azkar probatzeko ezintasunak aurrerapena moteltzen zuen, ikerlari berriei eremu horretan sartzea zailduz.

2015ean TF [3, 2] eta 2017an PyTorch [171, 172] sartu zirenean, ADa eskuragarriago bihurtu zen ikertzaileentzat, eskuzko konputazioaren diseinuetatik askatuz, eta garapen eta berrikuntza azkarragoak bultzatuz. NNetara bideratutako plataforma horien agerpenari esker, zeinak ADetan oinarrituta baitzeuden eta erabilerrazak baitziren, 2010eko hamarkadaren amaieratik aurrera PDE-en ebazpenaz arduratutako komunitateetan NNetan oinarritutako bideak ikertzeko

[13]Ikusi [170] oinarri-erradial NNen UAT aurrekari baterako.
[14]Ingelsetik, *Automatic Differentiation (AD)*.

interesa piztu zen.

Zehazki, *fisikatik informatutako neurona-sareak (PINNak)*[15] PDE-en ebazpenaz arduratutako komunitatean NNak erabiltzearen aurrekari garrantzitsuenetakotzat har daitezke, bereziki azken bost urteetan. PINNen lehen bertsioa bi zatitan argitaratu zen 2017an [180, 181]. Han, egileek datuetan oinarritutako alternatibak proposatu zituzten PDEak ebazteko ([180]) edo aurkitzeko ([181]). "Ebaztea" eta "aurkitzea" bata bestearengandik bereizteko, pentsa dezagun koefiziente (parametro) jakin batzuen mende dagoen operadore diferentzial bat duen PDE batean. Alde batetik, koefiziente guztien balioak ezagutzen baditugu, PDEan inplikatutako lege diferentzialerako sarbide osoa eskuragarri izango dugu. Beraz, lehen aipatutako "lehen laneko" [63][16] terminoetara murritz gaitezke PDEaren soluzioa hurbiltzeko (gogoratu 1.4.1. ataleko lehenengo zatia). Bestalde, koefiziente batzuk ezezagunak badira, baina PDEaren soluzioaren lagin batzuk izan baditzakegu espazio-denborako hainbat puntutan, ikaskuntza-eskema gainbegiratu bat ezar dezakegu, non NN bat (PDEaren soluzioa irudikatzen duena) eta parametro entrenagarri gehigarri batzuk (PDEaren koefiziente ezezagunak adierazten dituztenak) PDEa eta etiketatutako datuak asetzeko murrizketarekin entrenatzen baitira.

PINN terminologiarekin jarraituz, formulazio bariazionalak jorratzen dituzten lanak aurkitzen ditugu, *PINN (hp-)bariazionalak ((hp-)VPINNak)* deituak [115, 116, 188]; kontserbazio-legekoak, *PINN kontserbatzaileak (cPINNak)* deituak [107]; zatikizko ordena duten PDE-entzako, *PINN zatikidunak (fPINNak)* deituak [168]; eta datu zaratatsuetarako proposamen bayesiarrak, *PINN bayesiarrak (B-PINNak)* deituak [229]. PINNek mundu errealeko fenomenoetan dituzten aplikazioen artean, honako hauek nabarmentzen ditugu: [149]-k abiadura handiko fluxuetarako, [157, 99]-k potentzia-sistemetarako, [200]-ek ultrasoinuen bidezko kuantifikazio ez-suntsitzailerako, [43]-k beroa transferitzeko eta [42, 111]-k fluidoen mekanikarako. [53, 126, 35]-etara jotzen dugu PINNetan lotutako berrikuspen bibliografiko zehatzagoetarako.

Bestela, PDE-en NNetan oinarritutako bereizmenaren helburuari jarraituz, *deep* izena hartzen duten lanak ere proposatu dira. Batzuk aipatzekotan: [68, 69]-k *Deep Ritz metodoa (DRMa)* aurkezten dute, PDE-en bariazional-formulazioan inplikatutako Ritz-en energia-funtzioa minimizatzen duena ([139, 65, 135, 217] lanak aipatzen ditugu analisi eta hedapen gehigarrietarako); [201]-ek *Deep Galerkin*

[15]Ingelesez, *Physics-Informed Neural Networks (PINNs)*.

[16]Bitxia bada ere, ez [180, 181]-ek ezta antzeko egiletza-berrikuspenek ere, esaterako [182, 114], ez dute aipatzen [63]-ren lana. Berrikuspen orokorragoak kontsultatu behar ditugu (egiletzarekiko independenteak) erlazio hori aurkitzeko. Adibidez, [53]-k baieztatzen du (itzulita): "Ikaskuntza automatikoko algoritmo batean aurretiazko ezagutza txertatzearen kontzeptua ez da guztiz berria. Izan ere, Dissanayake eta Phan-Thien-ren [63] lana lehen PINNetakotzat har daiteke".

metodoa (DGMa) proposatzen du ([49, 131, 7, 198] hedapen, aplikazio eta konparazio gehigarrietarako); [45]-ek *Deep Least-Squares (DLS) metodoa* proposatzen du ([44, 137, 161] hedapen gehigarrietarako); eta [212]-k *Deep Fourier Residual (DFR) metodo* bat aurkezten du, [211]-n Maxwell denbora-armoniko ekuazioaren duela gutxiko hedapenarekin.

PDE parametrikoen bereizmenari buruzko zenbait lan berri ere aurki ditzakegu. Garrantzitsuenen artean: [134]-k maila anitzeko grafoen NN bat proposatzen du; [133]-k Fourierren espazioan parametrizazio bat erabiltzen du problema parametrikoetarako egokia den arkitektura adierazkor eta eraginkorrarekin; [34]-k NNak modeloen murrizketarekin konbinatzen ditu; [117]-k NNak erabiltzen ditu magnitude fisiko interesgarria parametrizatzeko sarrera-koefizienteen arabera; [122]-k PDE parametrikoetarako NNen bidezko mapa parametrikoen hurbilgarritasuna teorikoki aztertzen du; eta [173]-k PINNak bertsio parametriko batera hedatzen ditu, "metalearning" ikuspuntu deritzonaren bidez.

1.5. Ekarpen nagusiak

Tesi honek hiru ekarpen nagusi proposatzen ditu PDEak NNen bidez ebazteko: (i) elementu finituetan oinarritutako ikaskuntza sakoneko ebazle bat PDE parametrikoetarako; (ii) hondarren minimizazio esparru orokor bat, Ritz minimizazio bikoitzeko eskema batean oinarritua; eta (iii) zenbakizko memorian oinarritutako integrazio-estrategia bat entrenamenduan zehar integrazio-errorea modu eraginkorrean murrizteko.

1.5.1. Deep elementu finituen metodoa

Alderantzizko problemak ebaztea balio handikoa da gure gizartearentzat [228, 123] ingeniaritzako hainbat diziplinatan duen aplikagarritasunagatik, hala nola, irudi prosezakuntzan [33], elektromagnetismoan [10], ebaluazio ez-suntsitzaileetan [166] eta geofisikan [189], batzuk aipatzearren. Horiek ebazteko hainbat metodo daude, gradienteetan eta estatistikoetan oinarritutakoak barne [210, 16]. Metodo tradizional horiek alderantzizko soluzioa puntualki ebaluatu ohi dute (hau da, neurri multzo jakin baterako), baina oso gutxitan ematen dute alderantzizko operadore baten irudikapen globala. Hori gainditzeko eta alderantzizko operadore oso bat hurbiltzeko, NNak erabil daitezke (ikus, adibidez, [104, 231, 167, 196, 11, 197]), hurbilketa unibertsalaren propietateagatik.

Alderantzizko problemak ebazteko NNen entrenamendua *prozesamendu grafikoko unitateetan (GPUetan)*[17] egiten da, batez ere, datu-base handiekin edo NNen arkitektura konplexuekin lan egiten denean. Egoera horietan, komeni da

[17]Ingelesetik, *Graphics Processing Unit (GPU)*.

PDE parametrikoak ebazteko gai den metodo eraginkor bat izatea, GPUan inplementatua, NN moduan, alderantzizko arazoa entrenatzean parametro hautagaien gainean azkar iteratzeko. Baldintza horietan, alderantzizko eredua entrenatzeko bi ikuspegi nagusi daude: (i) aurreranzko eredu parametrikoa aurrez entrenatzea, ondoren denbora errealean ebazle/ebaluatzaile gisa erabiltzeko, edo (ii) aurreranzko eta alderantzizko ereduak aldi berean entrenatzea (auto) kodifikatzaile-dekodifikatzaile eskema batean [77].

Tesi honen lehen ekarpen nagusi gisa, **NNetan oinarritutako eredu bat proposatzen dugu, FEMean inspiratutako aurreranzko ebazle parametriko gisa jarduten duena**. Zehazki, gure NNa arkitektura elementu-finituen diskretizazio-sarearen mende dago, dinamikaren eragina imitatuz sare-finketak aplikatzean [18, 6, 208].

Proposatutako metodoaren, *Deep elementu finituen metodoa (DeepFEMa)*[18] edo *elementu finituen metodo sakona* deitua, aurkezpena eta inplementazioa *dimentsio bakarreko (1Dko)*[19] problemetara mugatzen da, zatika-linealak diren hurbilketak eta finketa uniformeak erabiliz. Dimentsio handiagoko problemetarako, zatika polinomikoak diren maila handiagoko hurbilketetarako eta/edo diskretizazio-sare adaptatiboetarako hedapena berehalakoa da, baina lan honen irismenetik haratago dago. DeepFEMa problema positibo-definituetan eta mugagabeetan probatzen dugu, parametro konstante eta zatika-konstanteekin. Oro har, esperimentuek hurbilketa onak erakusten dituzte; hala ere, batzuetan, optimizatzaileak eta ganbiltasun-mugek zehaztasun handiko soluzioak lortzea eragozten digute.

1.5.2. Deep Ritz metodo bikoitza

Ikuspegi abstraktutik, formulazio bariazionalek [155] (non, adibidez, FEMa oinarritzen baita) BVPak ekuazio funtzionaletan bihurtzen dituzte honako era honetan:

$$Bu = l \in \mathbb{V}', \tag{1.4}$$

non $\mathbb{U}$ eta $\mathbb{V}$ proba- eta test-espazioak (normadunak) deitzen baitira, hurrenez hurren; $\mathbb{V}'$ espazioak $\mathbb{V}$ espazioaren duala adierazten du; eta $B : \mathbb{U} \longrightarrow \mathbb{V}'$ eta $l \in \mathbb{V}'$ jatorrizko operadore diferentzialaren eta iturburu terminoen “analogo bariazionalak” dira, hurrenez hurren. [58, 150. orrialdea, (2.1)–(2.3) ekuazioak] aipatzen dugu xehetasun gehiagorako. Zehazki, honako baieztapena azpimarratzen dugu (itzulita): “problema bariazionalen izaera dagokion operadoreak espazio dual batean balioak hartzean datza”.

Formulazio abstraktu horren pean, *hondar-minimizazioa* (1.4) birformulazio

[18]Ingelesetik, *Deep Finite Element Method (DeepFEM)*.
[19]Ingelesetik, *one-dimensional (1D)*.

gisa sortzen da, honako era honetan:

$$\min_{u\in\mathbb{U}} \|Bu - l\|_{\mathbb{V}'}, \tag{1.5}$$

non norma dualaren erabilera $\|\cdot\|_{\mathbb{V}'}$ ezinbestekoa baita, $Bu - l$ espazio dualean ($\mathbb{V}'$-n) balioak hartzen dituelako [58, 151 orrialdea, (2.5) ekuazioa].

Norma dualen gorenaren (ingelesez, *supremum*) definizioari jarraituz, proba- eta test-espazioen gaineko aulki-puntuaren arazoarekin (min-max[20]) amaitzen dugu:

$$\min_{u\in\mathbb{U}} \max_{v\in\mathbb{V}\setminus\{0\}} \frac{\langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}}{\|v\|_{\mathbb{V}}}. \tag{1.6}$$

Hemen, $\langle\cdot,\cdot\rangle_{\mathbb{V}'\times\mathbb{V}}$-k parekatze duala adierazten du eta $\|\cdot\|_{\mathbb{V}}$-k test-espazioan dagoen norma.

NNen testuinguruan, [232, 20]-k hondar-minimizazioa bi NN erabiliz egitea proposatu zuten, proba- eta test-funtzioak irudikatzeko min-max optimizazio-estrategian (1.6). Egileek, *sare aurkari ahul (WAN)*[21] deitu zioten metodoari. Jatorriz, [232, 20]-k saiakuntzan arau azpioptimotzat jo bazituzten ere (L^2-norma aukeratu zuten H^1-norma egokiaren ordez), eragozpen nagusia da min-max ikuspegi horren azpian dagoen ezegonkortasuna, proba minimizazio iterazio bakoitzean inplikatutako test unitarioaren maximizatzailearen bilaketan (hau 4. kapituluan erakutsiko dugu).

Aurreko ezegonkortasuna gainditzeko, **NNak erabiltzen dituen hondar-minimizazio metodo orokor bat proposatzen dugu**, *Deep Ritz metodo bikoitza (D²RM)*[22] edo *Ritz metodo bikoitza eta sakona* deitua, WANen antzeko ideia jarraitzen duena eta bi NNak erabiltzen dituena proba- eta test-funtzioetan abiarazpen iteratibo batean, baina Ritz minimizazio-formulazio egonkor batean oinarrituta [185]. Izan ere, D^{2}RMa WANen metodoaren bertsio egonkor gisa interpreta daiteke, edo DRMaren orokortze gisa problema ez-simetrikoetarako edo zehaztugabeetarako [69] (berrikus 1.4.2. atala).

WANen, DRMaren eta D^{2}RMaren arteko erlazioari jarraituz, hiru metodo horiek probatu eta alderatzen ditugu difusio- eta konbekzio-problema batzuetan, D^{2}RMaren orokortzeko gaitasuna eta egonkortasuna erakusteko (beste biekin alderatuta), kontuan hartutako NNak hurbiltzeko ezaugarrietara eta optimizatzaileen entrenamendu-gaitasunera arte.

[20]Hilbert espazioetan, non aurrerago mugatuko garen, gorena lortzen da eta, beraz, maximo batekin ordezka daiteke.

[21]Ingelesetik, *Weak Adversarial Networks (WANs)*.

[22]Ingelesetik, *Deep Double Ritz Method (D^{2}RM)*.

1.5.3. Memorian oinarritutako Monte Carlo integrazioa

BVP bat NNekin ebaztea, oro har, helburu funtzio baten minimizazio gisa deskribatzen da, NN bat integratzailean sartzen duen integral definitu baten forman —adibidez, gogoratu (1.3) edo (1.6)—. Beraz, zenbakizko hurbilketaren edo koadratura erregelaren kalitatea kritikoa da. Entrenamendu osoan zehar zenbakizko integraziorako koadratura erregela finko bat kontuan hartzen badugu, litekeena da NNak nahi ez diren iragarpenak sortzea [186] *gaindoikuntza* (ingelesetik, *overfitting*) izeneko fenomeno ezagunaren ondorioz [92, 51, 77].

Gaindoikuntza saihesteko ohiko aukera bat *Monte Carlo (MC)* integrazioa [163, 128] erabiltzea da, izaera estokastikoa duelako. Gainera, sarerik gabeko egiturak geometria konplexuetara edo dimentsio handiko domeinuetara zuzenean eskalatzeko aukera erraza eskaintzen du. Zoritxarrez, MC integrazioak $\mathcal{O}(1/\sqrt{N})$ ordenako errorea eragiten du, non N integrazio-puntuen kopurua baita [163]. Integrazioaren zehaztasun-tasa txiki hori hamar mila edo ehun mila integrazio-punturen laginketak eginez gainditzen da normalean (entrenamendu-iterazio bakoitzean), entrenamendu-abiadura nabarmen murriztuz.

Tesi honen hirugarren ekarpen nagusi gisa, **memorian oinarritutako MC integrazio eskema bat proposatzen dugu PDEak ebazteko NNak erabiltzen direnean**. Hemen, memoriaren helburua da aurreko iterazioetan kalkulatutako zenbatespen integralak aprobetxatzea, egungo iterazioan zenbatespen integraletan laguntzeko. Eskema horrek integrazio-errorea gutxitzen du, entrenamenduan lagin desproportzionatuekin tratatzearen gainkarga konputazionalean erori gabe. Gainera, ikuspegi hori *gradiente jaitsiera estokastiko (SGD)*[23] optimizatzailera behar bezala eramatean, *momentum* metodoaren aldeko berrinterpretazio bat aurkitzen dugu [177].

1.6. Egitura

Tesi honen gainerakoa honela egituratuko da: 2. kapituluak FFNNak formalki aurkeztuko eta aztertuko ditu. 3., 4. eta 5. kapituluak tesiaren ekarpen nagusien garapen zehatzari buruzkoak dira, 1.5.1., 1.5.2. eta 1.5.3. ataletan laburtutakoak, hurrenez hurren. 6. kapituluak tesiaren ondorio nagusiak azalduko ditu eta etorkizuneko lanetan heldu beharreko erronken zirriborroa egingo du. Azkenik, 7. kapituluak doktorego-programaren baitan garatutako lorpen nagusiak deskribatuko ditu, argitaratutako lanen, emandako hitzaldien eta beste jarduera zientifiko garrantzitsu batzuen ekarpen zientifikoak aipatuz.

[23]Ingelesetik, *Stochastic Gradient Descent (SGD)*.

2. Feed-Forward Neural Networks

Herein, we formally introduce the *Feed-Forward Neural Networks (FFNNs)* that we will exhaustively exploit along Chapters 3, 4, and 5.

This chapter is organized as follows. Section 2.1 presents the architecture framework; Section 2.2 discusses continuum, parameterized, and discretized setups; Section 2.3 describes gradient-based training; and Section 2.4 illustrates the functioning of FFNNs with an easy-to-analyze case of study.

2.1. Architecture

In our case, a FFNN is a (vector-valued) function $\mathbf{u}_{\mathrm{NN}} : X \longrightarrow Y$ defined as follows:

$$\mathbf{y}_0(x) := x \in X \subset \mathbb{R}^{n_0}, \tag{2.1a}$$
$$\mathbf{y}_j(x) := \varphi(\mathbf{W}_j \mathbf{y}_{j-1} + \mathbf{b}_j) \in \mathbb{R}^{n_j}, \qquad 1 \leq j \leq K, \tag{2.1b}$$
$$\mathbf{u}_{\mathrm{NN}}(x) := \mathbf{W}\mathbf{y}_k \in Y \subset \mathbb{R}^{n_{K+1}}, \tag{2.1c}$$

where $\mathbf{W}_j\mathbf{y}_{j-1} + \mathbf{b}_j$ is an affine transformation determined by *weights* $\mathbf{W}_j \in \mathbb{R}^{n_j \times n_{j-1}}$ and *bias* $\mathbf{b}_j \in \mathbb{R}^{n_j}$ for $1 \leq j \leq K$, $\mathbf{W}\mathbf{y}_K$ is a linear transformation via weights $\mathbf{W} \in \mathbb{R}^{n_{K+1} \times n_K}$, and φ is a non-linear *activation function* that acts componentwise, i.e., for any $\mathbf{z} = (z^{(1)}, z^{(2)}, \dots, z^{(k)}) \in \mathbb{R}^k$ and any $k \geq 1$,

$$\varphi(\mathbf{z}) = \left(\varphi\left(z^{(1)}\right), \varphi\left(z^{(2)}\right), \dots, \varphi\left(z^{(k)}\right)\right). \tag{2.2}$$

We call *hidden layer* the mapping $\mathbf{y}_{j-1} \mapsto \mathbf{y}_j$ given in (2.1b), and the dimension of its output vector, n_j, is known as its *width*. Similarly, we have *input* and *output layers* in (2.1a) and (2.1c) with corresponding input and output dimensions n_0 and n_{K+1}, respectively. The *depth* of $\mathbf{u}_{\mathrm{NN}}$ is its number of hidden layers, K, that when reaching a significant value (e.g., more than three), the NN is typically referred to as a *Deep Neural Network (DNN)*. Figure 2.1 shows the graph of a fully-connected FFNN architecture and Figure 2.2 displays different widely used activation functions.

The parameterization of $\mathbf{u}_{\mathrm{NN}}$ is commonly given by all the elements involved in the affine transformations. Hence, to formally split the input domain versus

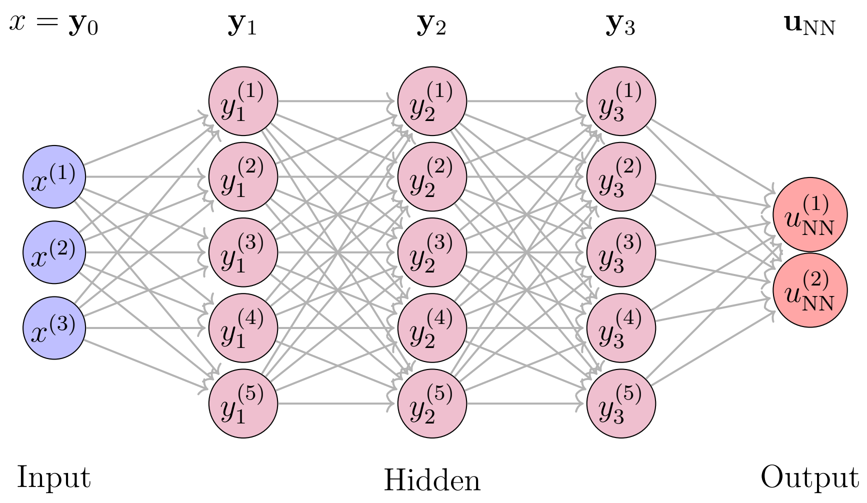


Figure 2.1: Graph of a fully-connected FFNN with a three-dimensional input, two-dimensional output, depth three, and hidden layers of width five.

the parameterization domain, we redefine the FFNN as follows:

$$\begin{aligned} \mathbf{u}_{\mathrm{NN}} : \quad X \quad \times \quad \Theta \quad &\longrightarrow \quad Y, \\ x \quad , \quad \theta \quad &\longmapsto \quad \mathbf{u}_{\mathrm{NN}}(x;\theta), \end{aligned} \tag{2.3a}$$

where

$$\theta := \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \dots, \mathbf{W}_K, \mathbf{b}_K, \mathbf{W}\} \tag{2.3b}$$

stands for the set of learnable parameters with domain

$$\Theta := \mathbb{R}^{n_1 \times n_0} \times \mathbb{R}^{n_1} \times \mathbb{R}^{n_2 \times n_1} \times \mathbb{R}^{n_2} \times \cdots \times \mathbb{R}^{n_{K+1} \times n_K} \times \mathbb{R}^{n_{K+1}} \times \mathbb{R}^{n_{K+1}}. \tag{2.3c}$$

The functions arising from this parameterization, also known as *realizations*, belong to a space that, in general, is not a vector space but a *manifold* [174]. Following the above notation, we denote it by

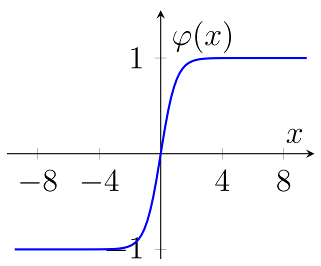
$$\mathbb{U}_{\mathrm{NN}} := \{\mathbf{u}_{\mathrm{NN}}(\,\cdot\,;\theta) : X \longrightarrow Y\}_{\theta \in \Theta}, \tag{2.4}$$

and by $\Phi_{\mathrm{NN}} : \Theta \longrightarrow \mathbb{U}_{\mathrm{NN}}$ the *realization map* that relates each configuration of parameters with the corresponding realization for the given NN architecture, i.e.,

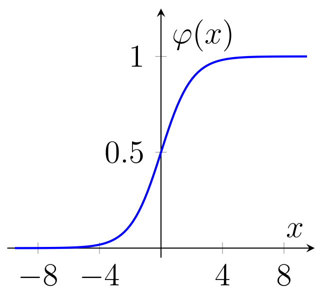
$$\Phi_{\mathrm{NN}}(\theta) := \mathbf{u}_{\mathrm{NN}}(\,\cdot\,;\theta) : X \longrightarrow Y, \qquad \theta \in \Theta. \tag{2.5}$$

We will drop the boldface notation when the NN is specified as a scalar-valued function, i.e., we will write u_{NN} instead of $\mathbf{u}_{\mathrm{NN}}$ whenever $n_{K+1} = 1$.

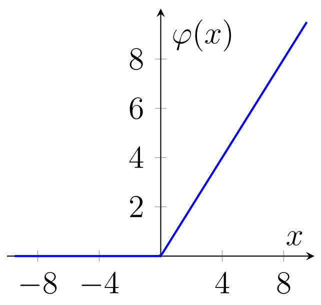
The set of realizations produced by a given FFNN defined as in (2.3) could be interpreted as a family of finite-dimensional vector spaces. See Appendix 2.A for details.



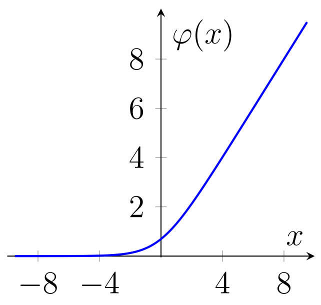
(a) $\varphi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(b) $\varphi(x) = \sigma(x) = \frac{e^x}{e^x + 1}$



(c) $\varphi(x) = \text{ReLU}(x) = \max\{0, x\}$



(d) $\varphi(x) = \text{softplus}(x) = \ln(1 + e^x)$

Figure 2.2: Some usual activation functions appearance: (a) sigmoid, (b) hyperbolic tangent, (c) rectified linear unit, and (d) softplus.

2.2. Continuum, parameterized, and discretized setups

Although NNs are often venerated for their tremendous approximation capabilities, they also possess some noteworthy topological undesirable properties. For example, given a NN architecture, the set of realizations is generally non-closed and non-convex [174, 40]. This typically translates into possibly being unable to establish an "optimal" realization (that is better than any other) or that multiple ones may exist. This is a substantial departure from the usual approach where the optimal approximation candidate is typically well-defined as the projection onto a pre-established finite-dimensional vector subspace. Aware of these ill-posed omens, we formally present the approximation task via NNs as follows.

Let $u^* : X \longrightarrow Y$ be a function that belongs to a certain suitable space of functions $\mathbb{U}$, e.g., an infinite-dimensional vector space, and let $\mathcal{F} : \mathbb{U} \longrightarrow \mathbb{R}$ be an *objective function* that characterizes u^* as its well-defined minimizer, i.e.,

$$u^* = \arg\min_{u \in \mathbb{U}} \mathcal{F}(u). \tag{2.6}$$

Let $u_{\text{NN}} : X \times \Theta \longrightarrow Y$ be a NN architecture according to definition (2.3), and assume that the corresponding set of realizations $\mathbb{U}_{\text{NN}}$ is embedded in $\mathbb{U}$. Then, we define *an optimal approximation of u^* via u_{NN}* as

$$u^*_{\text{NN}} \in \mathbb{U} \text{ such that } \mathcal{F}(u^*_{\text{NN}}) = \inf_{u_{\text{NN}} \in \mathbb{U}_{\text{NN}}} \mathcal{F}(u_{\text{NN}}). \tag{2.7}$$

Although u^*_{NN} may not belong to $\mathbb{U}_{\text{NN}}$, by assuming that both Φ_{NN} and $\mathcal{F}$ are continuous on Θ and $\mathbb{U}$, respectively, we obtain that both $\mathbb{U}_{\text{NN}}$ and $\mathcal{F}(\mathbb{U}_{\text{NN}})$ are (path-)connected. Then, there always exists a sequence of elements in $\mathbb{U}_{\text{NN}}$ that converges to the accumulation point u^*_{NN}. In other words, u^*_{NN} can be approximated with arbitrary precision by means of $\mathbb{U}_{\text{NN}}$, as well as $\mathcal{F}(u^*_{\text{NN}})$ by means of $\mathcal{F}(\mathbb{U}_{\text{NN}})$. Figure 2.3 illustrates this setup.

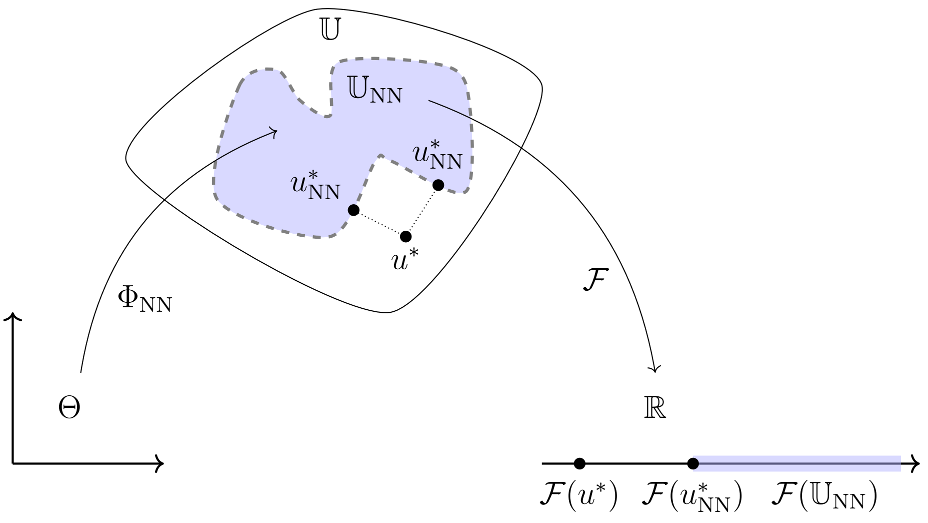


Figure 2.3: Sketch of the parameterization of a NN and its objective-function minimization formulation.

So far, we presented the *parameterization* in NNs as an embedding of a finite-dimensional Euclidean space Θ into $\mathbb{U}$ via the realization mapping Φ_{NN}. Some authors might refer to this process as a *discretization* due to the utilization of a finite number of parameters to represent the approximating functions. However, to avoid confusion, we will avoid the latter term to distinguish it from the one concerning the input domain X that we discuss below.

Assume that the objective function possesses the following integral form:

$$\mathcal{F}(u) := \int_X I(u)(x)\, dx, \qquad u \in \mathbb{U}, \tag{2.8}$$

where $I(u)$ stands for the integrand that consists of transformations of u. Then, the natural restriction of $\mathcal{F}$ to $\mathbb{U}_{\text{NN}}$ is by means of the parameterization Φ_{NN}, i.e.,

$$\mathcal{F}_{\text{NN}}(\Phi_{\text{NN}}(\theta)) = \int_X I(\Phi_{\text{NN}}(\theta))(x)\ dx, \qquad \theta \in \Theta. \tag{2.9}$$

To address the challenges associated with the exact integration of NN, we consider

$$\mathcal{L} : \Theta \times X^N \longrightarrow \mathbb{R} \tag{2.10a}$$

defined by

$$\mathcal{L}(\theta; x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N} \omega_i\ I(\Phi_{\text{NN}}(\theta))(x_i), \tag{2.10b}$$

where $x_i \in X$ and $\omega_i > 0$ are conveniently selected integration points and weights, respectively, so as to resemble a suitable quadrature rule to approximate $\mathcal{F} \circ \Phi_{\text{NN}}$, i.e.,

$$\mathcal{F}(\Phi_{\text{NN}}(\theta)) \approx \mathcal{L}(\theta; x_1, x_2, \ldots, x_N), \qquad \theta \in \Theta, x_i \in X, 1 \leq i \leq N. \tag{2.10c}$$

Thus, we obtain a *parameterized* and *discretized* representative $\mathcal{L}$ of the objective function $\mathcal{F}|_{\mathbb{U}_{\text{NN}}}$ intended to be computable. Moreover, we may endow a *stochastic* nature to (2.10) by sampling the integration points $\{x_i\}_{i=1}^N \subset X$ according to a random distribution every time we estimate the value of the integral. Example 2.1 shows a widely used stochastic integral approximation approach when using NNs.

Example 2.1 (Monte Carlo integration)**.** For $\text{Vol}(X) := \int_X 1dx < \infty$ and $\{x_i\}_{i=1}^N \subset X$ following a random uniform distribution, *Monte Carlo (MC) integration* consists of selecting $\omega_i = \text{Vol}(X)/N$ in (2.10) for all $1 \leq i \leq N$, i.e.,

$$\mathcal{F}(\Phi_{\text{NN}}(\theta)) = \int_X I(\Phi_{\text{NN}}(\theta))(x)\ dx, \qquad \theta \in \Theta, \tag{2.11a}$$

$$\approx \frac{\text{Vol}(X)}{N} \sum_{i=1}^{N} I(\Phi_{\text{NN}}(\theta))(x_i) = \mathcal{L}(\theta; \{x_i\}_{i=1}^N), \qquad \theta \in \Theta. \tag{2.11b}$$

Henceforth, we will refer to the parameterized, discretized, and possibly stochastic rendition of the objective function as the *loss function*. In this way, we distinguish the continuum-level functional that characterizes our minimization formulation, $\mathcal{F}$, from its “computationally feasible” counterpart, $\mathcal{L}$.

In data science, it is common to directly encounter the minimization formulation in a discretized and stochastic form. Example 2.2 presents a so-typical supervised learning approach with NNs.

Example 2.2 (Supervised learning with NNs)**.** Let $D = \{x_i, y_i\}_{i\geq 1} \subset X \times Y$ be an available labeled large database, and let u_{NN} be a given NN model. Then, the approximation task is typically presented as the minimization of

$$\mathcal{L}(\theta; \{x_i, y_i\}_{i=1}^N) = \frac{1}{N} \sum_{i=1}^{N} \|u_{\text{NN}}(x_i; \theta) - y_i\|, \tag{2.12}$$

where $\|\cdot\|$ is a pre-established discrete norm and $\{x_i, y_i\}_{i=1}^N$ is a stochastically selected subsample from D, usually called *batch*, such that $N << |D| < \infty$.

This loss-function-level presentation could be viewed as a Monte Carlo approximation of the continuum-level objective function

$$\mathcal{F}(u) = \frac{1}{\text{Vol}(X)} \int_X \|u_{\text{NN}}(x; \theta) - u^*(x)\| \, dx, \tag{2.13}$$

where u^* stands for the continuum-level function from which we extracted the labeled database, i.e., $y_i = u^*(x_i)$.

Overfitting [92, 51, 77] is a common issue during training, which makes the NN learn to perform exceptionally well on the training data but struggles to generalize to new (unseen) data. In our integration context, this usually means that the NN focuses on satisfying the quadrature requirement only at the integration points (the training data) but possibly misbehaves elsewhere. In [186], the authors show situations where integrating the NN with few fixed points leads to greedy behavior during training, producing extremely undesirable solutions. To overcome this, MC integration is typically utilized due to its stochastic nature. Unfortunately, MC integration demands an immense sample size to avoid making large integration errors (recall Section A.5.3/1.5.3).

2.3. Gradient-based training

To carry out the minimization, we adopt first-order gradient-based methods. Here, the term "first order" specifies that we *only* calculate gradients with respect to the learnable parameters (and not higher-order derivatives).

The cost of computing the gradient of a *scalar-valued function* is known to be bounded by the cost of evaluating the function itself times a constant when using a usual backpropagation algorithm via Automatic Differentiation (AD) [82, 27, 150]. Indeed, such a constant is known to be smaller than or equal to three under specific theoretical conditions (Baur-Strassen Theorem [25]). Hence, the computational cost of calculating the gradient of the objective/loss function is comparable to simply evaluating it.

However, this fact cannot be extrapolated to *vector-valued functions*. There, the cost of computing its gradient behaves as the cost of computing the componentwise gradients, i.e., it is comparable to the cost of evaluating the vector function multiplied by the number of its (scalar-valued) output components. Then, second-order minimization methods for NNs have a cost proportional to evaluating the loss/objective function times the number of learnable parameters. Since NNs typically follow an over-parameterized design to exploit their approximation capacities, second or higher-order optimization methods become extremely expensive.

In what follows, we present the gradient-descent method that serves as the foundation of the majority of currently employed optimizers when training NNs. We first introduce this method at a continuum level to later deduce its parameterized, discretized, and stochastic counterpart.

2.3.1. Continuum-level Gradient-Descent method

Let $\mathbb{U}$ be a Hilbert space and, in addition to the well-posedness, assume that $\mathcal{F} : \mathbb{U} \longrightarrow \mathbb{R}$ is *convex*, i.e., for $u, w \in \mathbb{U}$ and $0 \leq \lambda \leq 1$, it satisfies

$$\mathcal{F}(\lambda u + (1-\lambda)w) \leq \lambda \mathcal{F}(u) + (1-\lambda)\mathcal{F}(w), \tag{2.14}$$

and *sufficiently differentiable*. Then, the gradient[1] operator $\nabla\mathcal{F}$ is monotonically increasing (Kachurovskii's Theorem [112]), and thus provides an ideal framework to carry out a continuum-level gradient-descent minimization [204] defined as follows:

$$u_t = u_{t-1} - \lambda \nabla \mathcal{F}(u_{t-1}), \tag{2.15}$$

where $\lambda > 0$ is the *learning rate* and $u_t \in \mathbb{U}$ is the function at the t^{th} iteration. Intuitively, the gradient-descent method begins selecting an initial candidate $u^{(0)} \in \mathbb{U}$ and, iteratively, follows steps against the gradient, which by an appropriate (adaptive) control of the size of the learning rate, it is well known that we can approach u^* as much as desired [32, 204]. Figure 2.4 illustrates a graphical performance of the gradient-descent method on a convex scenario, and Example 2.3 briefly analyzes a simple continuum-level case of study.

Example 2.3. Let $\|\cdot\|_{\mathbb{U}}$ denote the norm induced from the inner product of the Hilbert space $\mathbb{U}$. Consider the objective function defined by $\mathcal{F}(u) := \|u\|_{\mathbb{U}}$. Then, $\mathcal{F}$ is convex on $\mathbb{U}$ and Fréchet differentiable on $\mathbb{U} \setminus \{0\}$ with gradient

[1] In Hilbert spaces, we refer to the operator defined via the Riesz representatives of the Fréchet derivative at each point, i.e., at a Fréchet differentiable point $u \in \mathbb{U}$, $\nabla\mathcal{F}(u)$ denotes the unique element in $\mathbb{U}$ that satisfies $D_u\mathcal{F}(w) = (w, \nabla\mathcal{F}(u))_{\mathbb{U}}$ for all $w \in \mathbb{U}$, where $D_u\mathcal{F}$ denotes the usual Fréchet derivative at $u \in \mathbb{U}$ and $(\cdot,\cdot)_{\mathbb{U}}$ stands for the inner product in $\mathbb{U}$.

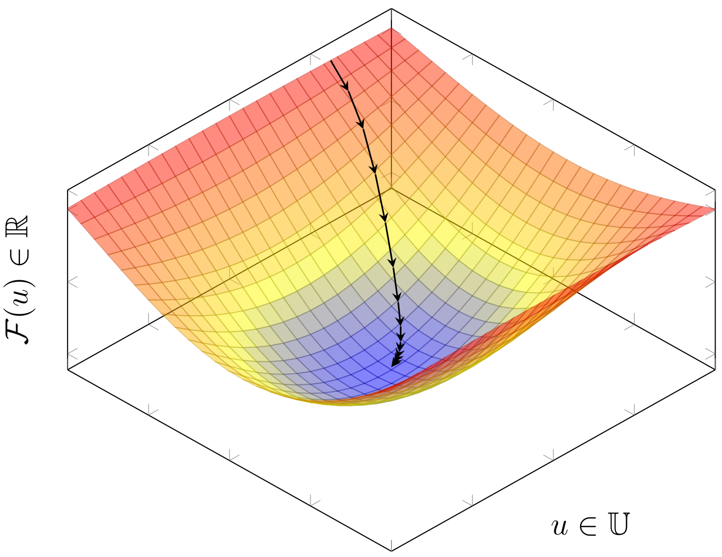


Figure 2.4: Gradient-descent performance sketch on a convex scenario.

$\nabla \mathcal{F}(u) = \frac{u}{\|u\|_{\mathbb{U}}}$. In consequence, given an initial candidate $u_0 \in \mathbb{U} \setminus \{0\}$, the continuum-level gradient-descent iterative method described in (2.15) becomes

$$u_t = u_{t-1} - \lambda \frac{u_{t-1}}{\|u_{t-1}\|_{\mathbb{U}}} = \left(1 - \frac{\lambda}{\|u_{t-1}\|_{\mathbb{U}}}\right) u_{t-1}. \tag{2.16}$$

Selecting $0 < \lambda = \lambda(t) \leq \|u_{t-1}\|_{\mathbb{U}}$, the iterative method converges to $0 \in \mathbb{U}$, which is the global minimum of $\mathcal{F}$ on $\mathbb{U}$. Indeed, if at any iteration we select $\lambda(t) = \|u_{t-1}\|_{\mathbb{U}}$, then we would reach the minimum at that moment.

2.3.2. Parameterized-level Gradient-Descent method

Because the continuum-level gradient-descent method is computationally intractable, to turn it into a practical method, we resort to the composition of the objective function with the available realization mapping,

$$\mathcal{F} \circ \Phi_{\text{NN}} : \Theta \longrightarrow \mathbb{R}, \tag{2.17}$$

and apply the corresponding gradient operator $\nabla(\mathcal{F} \circ \Phi_{\text{NN}})$. Because we are now in a finite-dimensional Euclidean domain, we identify the gradient with the vector of partial derivatives given by

$$\frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial \theta} := \left[\frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial \theta^{(i)}}\right]_{i=1}^{\dim(\Theta)}, \tag{2.18}$$

where $\theta^{(i)}$ denotes the i^{th} learnable parameter of θ. Consequently, the gradient-descent iterative method transletes into the domain of the set of learnable parameters as follows:

$$\theta_t = \theta_{t-1} - \lambda \frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial \theta}(\theta_{t-1}), \tag{2.19}$$

where $\theta_t \in \Theta$ stands for the set of learnable parameters at the t^{th} iteration.

While (2.19) can be seen as a parameterized-level version of (2.15), it is essential to note that some of the ideal conditions observed at the continuum level in the gradient-descent method, such as the convexity, do not necessarily satisfy at the parameterized level when dealing with NNs. Indeed, (2.19) merely aims to converge to parameters in Θ whose partial derivatives tend to vanish, which may lead to undesirable results (e.g., when falling in local minima or saddle points). This supposes another departure from the usual vector-space-based approach where the convexity condition of $\mathcal{F}$ is preserved in linear parameterizations (see Example 2.4).

Example 2.4 (Convexity preservation under linear transformations)**.** For simplicity, we first present the case of linear combinations. Let $\mathcal{F}$ be convex on $\mathbb{U}$ and consider Φ_{NN} defined by

$$\Phi_{\text{NN}}(\theta) = \sum_{i=1}^{\dim(\Theta)} \theta^{(i)} u^{(i)}, \qquad \theta = \{\theta^{(i)}\}_{i=1}^{\dim(\Theta)}, \tag{2.20}$$

where $\{u^{(i)}\}_{i=1}^{\dim(\Theta)} \subset \mathbb{U}$ is a pre-established set of functions. Then, $\mathbb{U}_{\text{NN}}$ is a vector space of dimension at most $\dim(\Theta)$ and $\mathcal{F} \circ \Phi_{\text{NN}}$ is convex on Θ.

Proof. The dimension of $\mathbb{U}_{\text{NN}}$ is the number of linearly independent functions in $\{u^{(i)}\}_{i=1}^{\dim(\Theta)} \subset \mathbb{U}$. To prove the convexity of $\mathcal{F} \circ \Phi_{\text{NN}}$, let $0 \leq \lambda \leq 1$ and $\theta, \vartheta \in \Theta$. By linearity, $\Phi_{\text{NN}}(\lambda\theta + (1-t)\vartheta) = \lambda\Phi_{\text{NN}}(\theta) + (1-\lambda)\Phi_{\text{NN}}(\vartheta)$. We obtain the desired result by applying $\mathcal{F}$ and the convexity definition. $\square$

Following a similar reasoning, it is straightforward to check that the convexity condition is also preserved when replacing the linear combination with a general linear transformation. In particular, the restriction of $\mathcal{F} \circ \Phi_{\text{NN}}$ to the learnable parameters of the output layer in a FFNN according to Section 2.1 is convex whenever $\mathcal{F}$ is convex.

2.3.3. Stochastic Gradient-Descent method

To avoid the computational challenges associated with the integral form of $\mathcal{F}$, we resort to the loss-function approximation introduced in (2.10). Then, the so-called *Stochastic Gradient Descent (SGD) optimizer* arises as

$$\theta_t = \theta_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial \theta}(\theta_{t-1}; \mathbf{x}_t), \tag{2.21}$$

where $\mathbf{x}_t \in X^N$ denotes the stochastically sampled set of integration points at the t^{th} iteration. Note that (2.21) introduces discrete dependencies and uncertainty during training compared to (2.19).

Multiple variants of stochastic gradient-based optimizers have been designed and exploited in recent years with the aim of improving many of the aforementioned poor conditions in which NNs are immersed. Here are some of the most popular today: SGD with momentum [179] and/or Nesterov acceleration [162], Adagrad [66], Adadelta [233], RMSprop [95], Adam/AdaMax [119], and Nadam [64]. We refer to [191] for an overview.

2.4. A case of study

To illustrate many of the (undesirable) properties pointed out so far, we discuss the following easy-to-analyze case of study.

At the continuum level, let $X := (-1,1)$, $Y := \mathbb{R}$, and

$$u^*(x) := \operatorname{sgn}(x) = \begin{cases} -1, & \text{if } x < 0, \\ 0, & \text{if } x = 0, \\ 1, & \text{if } x > 0. \end{cases} \tag{2.22}$$

Consider $\mathbb{U} := L^2(X) = \{u : X \longrightarrow \mathbb{R} : \int_X u^2 < \infty\}$ and let

$$\mathcal{F}(u) := \|u - u^*\|^2_{L^2(X)} = \int_X (u(x) - u^*(x))^2 \, dx, \tag{2.23}$$

whose minimum in $\mathbb{U}$ is zero and is attained only at u^*.

At the parameterization level, let $\Theta := \mathbb{R}^2$ and let $u_{\text{NN}} : X \times \Theta \longrightarrow \mathbb{R}$ be defined by

$$u_{\text{NN}}(x;\theta) := c \tanh(ax), \tag{2.24}$$

where $\theta = \{a, c\}$ denotes the set of learnable parameters. Then,

$$\mathbb{U}_{\text{NN}} = \Phi_{\text{NN}}(\Theta) = \{\Phi_{\text{NN}}(a,c) = c \tanh(ax) : X \longrightarrow Y\}_{a,c\in\mathbb{R}}, \tag{2.25}$$

and routinary calculations yield

$$(\mathcal{F} \circ \Phi_{\mathrm{NN}})(a, c) = 2(1 + c^2) - \frac{4c \log(\cosh(a)) + 2c^2 \tanh(a)}{a}, \tag{2.26}$$

$$\frac{\partial(\mathcal{F} \circ \Phi_{\mathrm{NN}})}{\partial a}(a, c) = \frac{2c\left(2 \log(\cosh(a)) + (c - 2a) \tanh(a) - ac(\operatorname{sech}^2(a)\right)}{a^2}, \tag{2.27}$$

$$\frac{\partial(\mathcal{F} \circ \Phi_{\mathrm{NN}})}{\partial c}(a, c) = \frac{4\left(ac - \log(\cosh(a)) - c \tanh(a)\right)}{a}, \tag{2.28}$$

for $a \neq 0$, $(\mathcal{F} \circ \Phi_{\mathrm{NN}})(0, c) = 2$, $\frac{\partial(\mathcal{F} \circ \Phi_{\mathrm{NN}})}{\partial c}(0, c) = -2c$, and $\frac{\partial(\mathcal{F} \circ \Phi_{\mathrm{NN}})}{\partial a}(0, c) = 0$. Figure 2.5 shows the graph of $\mathcal{F} \circ \Phi_{\mathrm{NN}}$, and Figure 2.6 shows the graphs of its partial derivatives.

The following statements expose a series of properties of this case of study:

1. The architecture u_{NN} posseses *arbitrary precision approximation capacity*, which means that $u^*_{\mathrm{NN}} = u^*$.

   Proof. Let $c = 1$. Then, it is straightforward to check that $(\mathcal{F} \circ \Phi_{\mathrm{NN}})(a, 1)$ converges to zero as $a \to \infty$, which implies that $\Phi_{\mathrm{NN}}(1, a)$ converges to u^* as $a \to \infty$. $\square$

2. The set of realizations $\mathbb{U}_{\mathrm{NN}}$ is *non-closed.*

   Proof. The optimal realization $u^*_{\mathrm{NN}} \in L^2(X) \setminus C^\infty(X)$ is unattainable because $\Phi_{\mathrm{NN}}(a, c) \in C^\infty(X)$ for all $a, c \in \mathbb{R}$. $\square$

3. The realization map Φ_{NN} is non-injective. In particular, the sequence of learnable parameters whose images through Φ_{NN} converge to u^*_{NN} is non-unique, although the limit realization u^*_{NN} is unique.

   Proof. The odd symmetry of $\tanh(x)$ implies $\Phi_{\mathrm{NN}}(a, c) = \Phi_{\mathrm{NN}}(-a, -c)$ for all $(a, c) \in \Theta$, which shows the non-injectivity. Then, for $c = 1$, we find distinct sequences $\{(a, 1)\}_{a \geq 1}$ and $\{(-a, -1)\}_{a \geq 1}$ of learnable parameters with distinct limits but whose images through Φ_{NN} converge to u^*_{NN} as $a \to \infty$. The uniqueness of u^*_{NN} follows from item 1. and the well-posedness of the objective-function minimization. $\square$

4. The set of realizations $\mathbb{U}_{\mathrm{NN}}$ is *non-convex.*

   Proof. We need to find $0 < \lambda < 1$ and $a_1, a_2, c_1, c_2 \in \mathbb{R}$ such that $f(x) = \lambda c_1 \tanh(a_1 x) + (1 - \lambda) c_2 \tanh(a_2 x)$ is unrealizable via Φ_{NN}, i.e., it is not of the form $\Phi_{\mathrm{NN}}(a, c) = u_{\mathrm{NN}}(x; a, c) = c \tanh(ax)$ for any $a, c \in \mathbb{R}$. Indeed, let $\lambda = 1/2$, $c_1 = c_2 = 2$, $a_1 = 2$, and $a_2 = -1$. Then, $f(x) = \tanh(2x) - \tanh(x)$, which is neither constant nor strict monotonic on X. Because $u_{\mathrm{NN}}(x; a, c)$ is either strict monotonic or constant on X, $f(x)$ is unrealizable via Φ_{NN}. $\square$

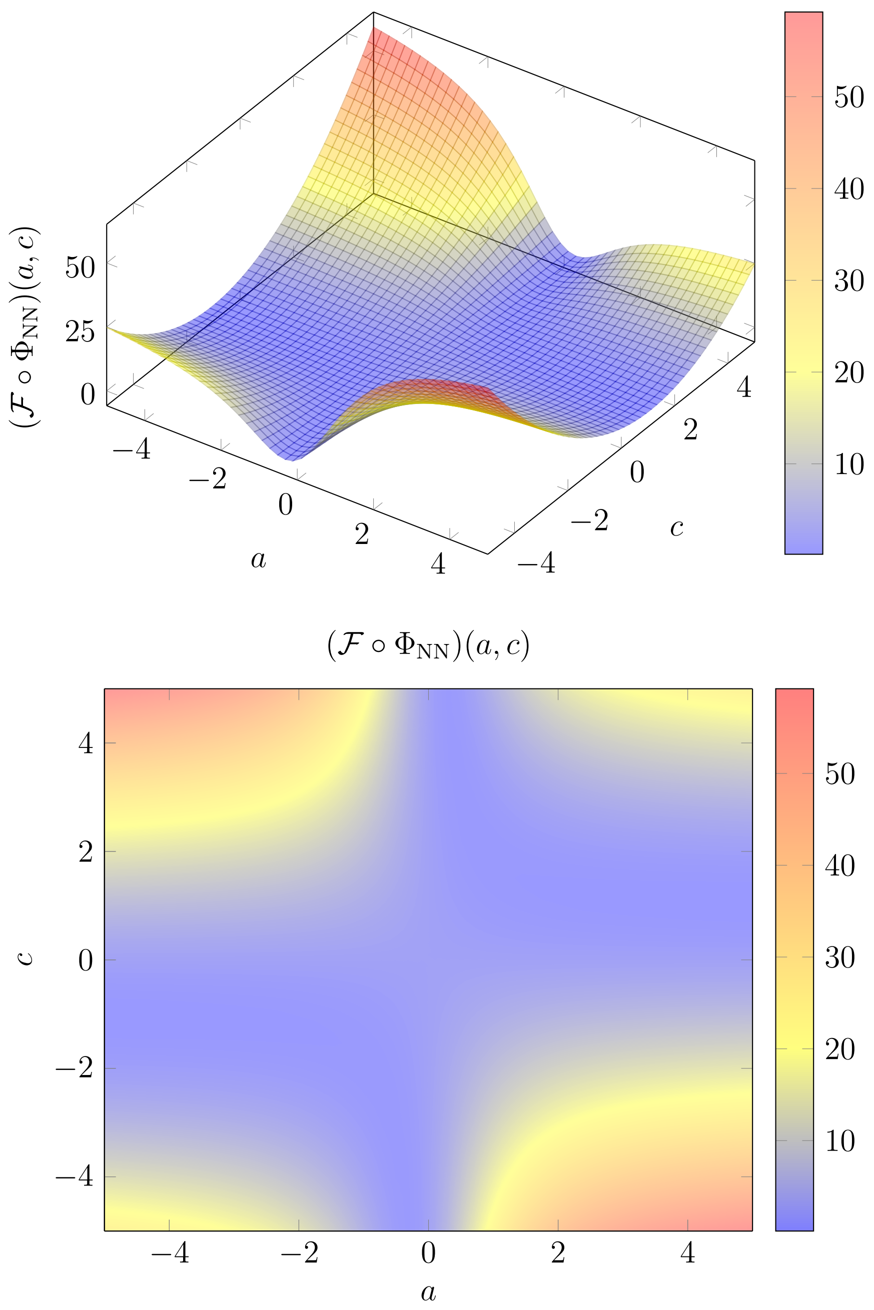
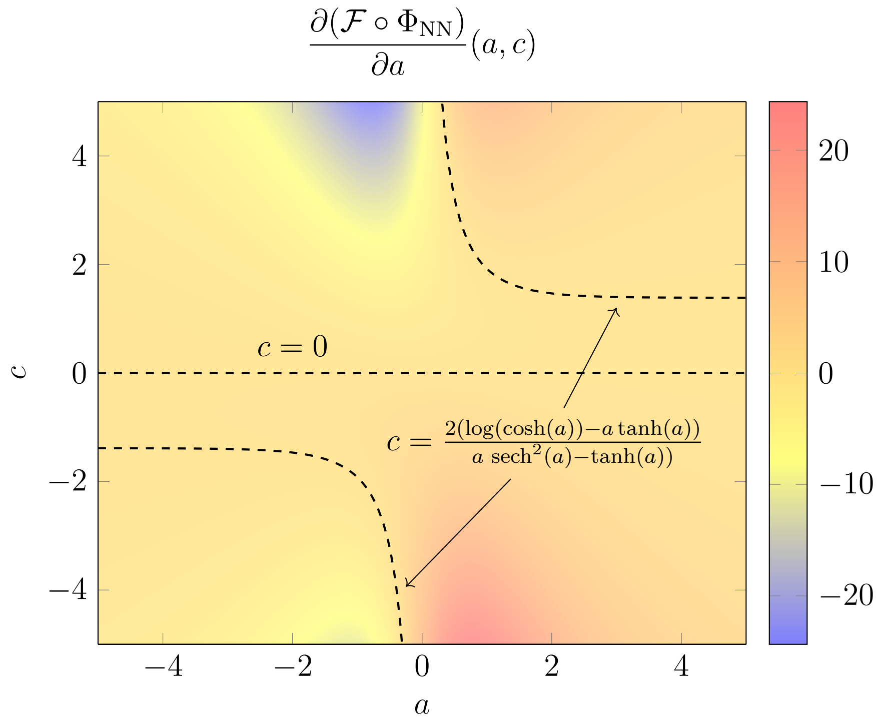


Figure 2.5: Graph of $\mathcal{F} \circ \Phi_{\text{NN}}$ for (2.23). Top panel: 3D view. Bottom panel: Top view.



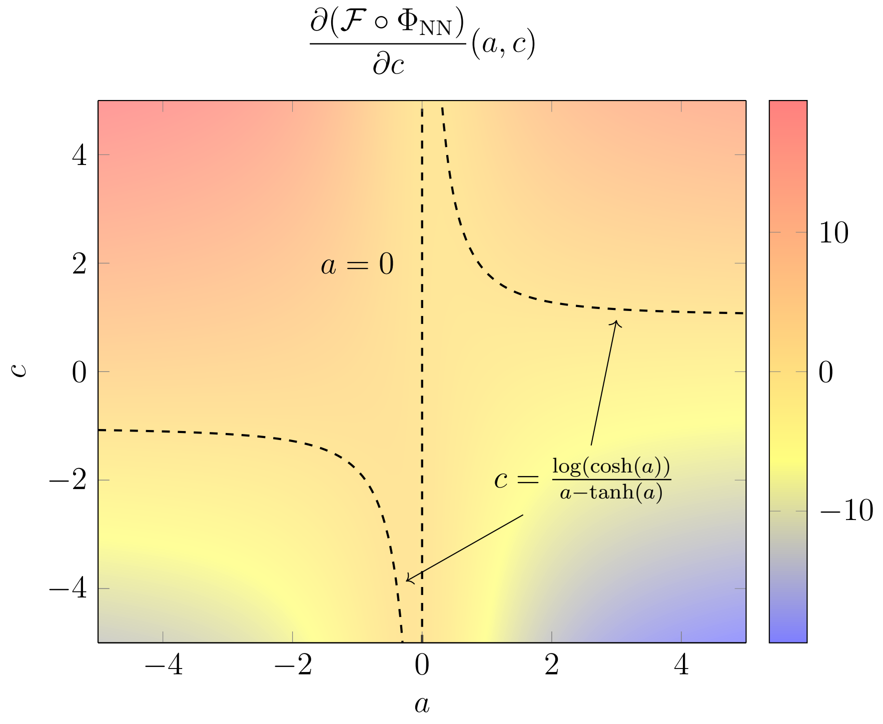


Figure 2.6: Graph of the partial derivatives of $\mathcal{F}\circ\Phi_{\mathrm{NN}}$ for (2.23). Dashed curves indicate vanishing regions.

5. $\mathcal{F}$ is *convex* on $\mathbb{U}$, but $\mathcal{F} \circ \Phi_{\text{NN}}$ is *non-convex* on Θ.

   Proof. Let $0 \leq \lambda \leq 1$ and $u_1, u_2 \in \mathbb{U}$. Then, the triangular inequality $\|\lambda(u_1 - u^*) + (1-\lambda)(u_2 - u^*)\|_{L^2(X)} \leq \lambda\|u_1 - u^*\|_{L^2(X)} + (1-\lambda)\|u_2 - u^*\|_{L^2(X)}$ implies $\mathcal{F}(\lambda u_1 + (1-\lambda)u_2) \leq \lambda\mathcal{F}(u_1) + (1-\lambda)\mathcal{F}(u_2)$, which shows the convexity of $\mathcal{F}$. To show the non-convexity of $\mathcal{F} \circ \Phi_{\text{NN}}$, fix any $c \neq 0$ and study the curvature of $f_c(a) := (\mathcal{F} \circ \Phi_{\text{NN}})(a, c)$—recall Figure 2.5. For example, $f_1''(a)$ is positive when $a > 0$ and negative when $a < -1$. □

6. The negative gradient field possesses undesirable attractive basin regions that may lead to poor results when performing the gradient-descent method.

   Proof. The only parameter configurations that lead to desirable realizations occur when $c = \pm 1$ and $a \to \pm\infty$, which implies $\frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial\{a,c\}}(a, \pm 1) \to (0, 0)$ as $a \to \pm\infty$. However, we can also obtain near-zero gradients in undesirable parameter configurations. Figure 2.7 shows poor gradient-descent performances that get trapped in near-zero-gradient regions. □

At the discretization level, we approximate $\mathcal{F} \circ \Phi_{\text{NN}}$ via a loss function that we expect to inherit similar conditions to above. Indeed, let $\mathbf{x} = \{x_i\}_{i=1}^N \subset X$. Then,

$$(\mathcal{F} \circ \Phi_{\text{NN}})(a, c) \approx \mathcal{L}(a, c; \mathbf{x}) = \sum_{i=1}^{N} \omega_i (c \tanh(a x_i) \pm 1)^2, \tag{2.29a}$$

$$\frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial a}(a, c) \approx \frac{\partial \mathcal{L}}{\partial a}(a, c; \mathbf{x}) = 2ac \sum_{i=1}^{N} \omega_i (c \tanh(a x_i) \pm 1) \operatorname{sech}^2(a x_i), \tag{2.29b}$$

$$\frac{\partial(\mathcal{F} \circ \Phi_{\text{NN}})}{\partial c}(a, c) \approx \frac{\partial \mathcal{L}}{\partial c}(a, c; \mathbf{x}) = 2 \sum_{i=1}^{N} \omega_i (c \tanh(a x_i) \pm 1) \tanh(a x_i), \tag{2.29c}$$

where ω_i denotes the integration weight related to the integration point x_i, and where the positive or negative sign depends on whether x_i is negative or positive, respectively.

When $c = \pm 1$ and $a \to \pm\infty$, we obtain the desired realizations convergence $\Phi_{\text{NN}}(a, c) \to u^*$ with corresponding vanishing of the gradient of the loss function

$$\frac{\partial \mathcal{L}}{\partial\{a, c\}}(a, \pm 1; \mathbf{x}) \to (0, 0) \text{ as } a \to \pm\infty, \quad \text{for any } \mathbf{x} \in X^N \text{ and } N > 0. \tag{2.30}$$

However, $\frac{\partial \mathcal{L}}{\partial\{a,c\}}(0, c; \mathbf{x}) = (0, 0)$ for any $c \in \mathbb{R}, \mathbf{x} \in X^N$ and $N > 0$, but $\Phi_{\text{NN}}(0, c) = 0 \neq u^*$ for all $c \in \mathbb{R}$. Hence, parameters of the form $(0, c) \in \Theta$ are undesirable critical points susceptible to trapping the learnable parameters during stochastic gradient-descent training.

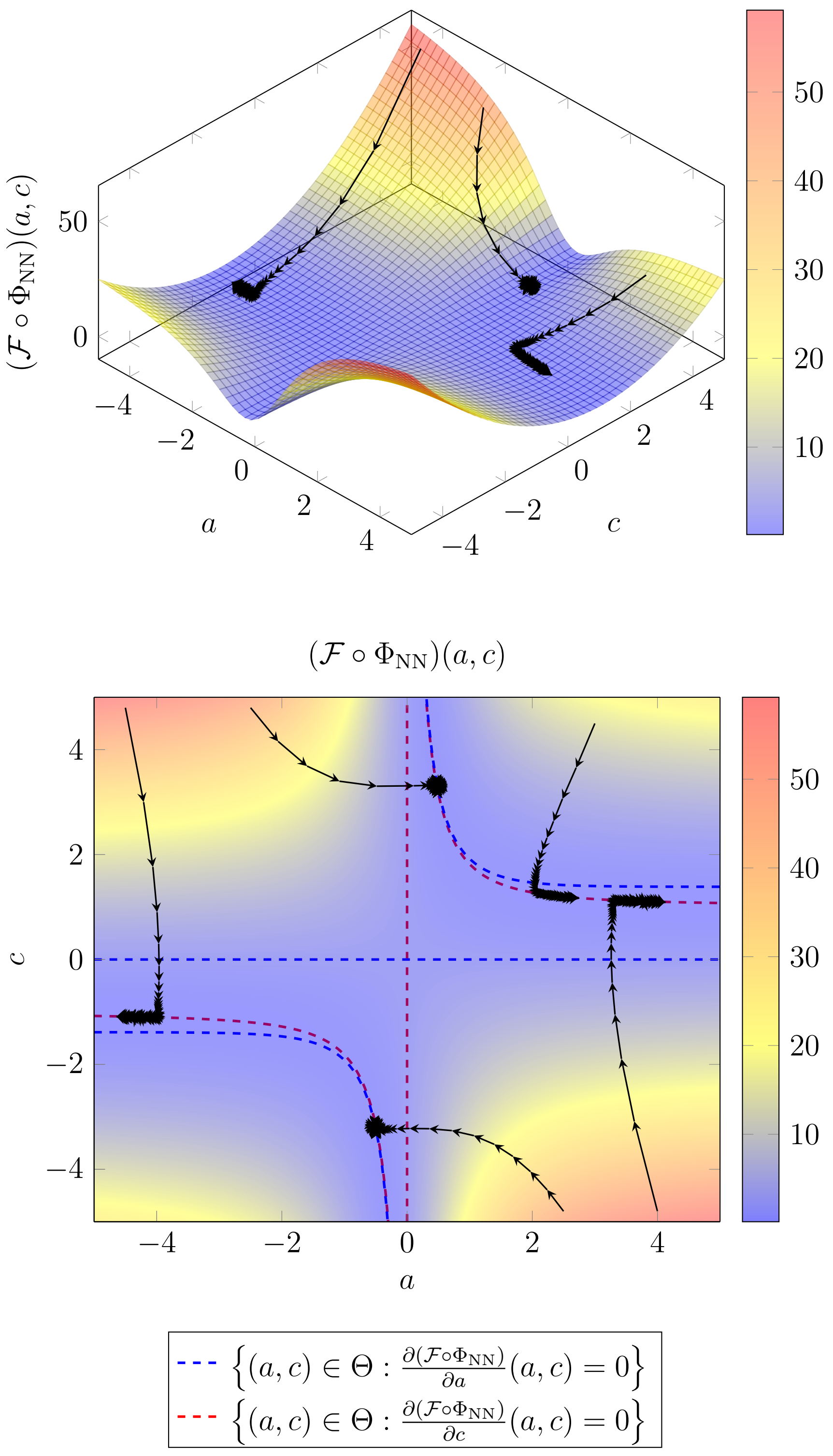


Figure 2.7: Different executions of the gradient-descent method for (2.23). We maintain constant learning rates in all cases and carry out 200 iterations. Top panel: 3D view. Bottom panel: Top view.

2.A. Interpretation of the set of realizations as a family of vector spaces

The set of realizations produced by a given FFNN defined as in (2.1) could be interpreted as a family of finite-dimensional vector spaces (up to a prescribed dimension). To this end, we argue guided under the following intuition: the collection of hidden layers is in charge of generating different finite-dimensional vector spaces, while the output layer is in charge of producing linear combinations over such spaces.

For simplicity, we first consider the scalar-valued case ($n_{K+1} = 1$), and then we extend it to the vector-valued case ($n_{K+1} > 1$).

Scalar-valued FFNN ($n_{K+1} = 1$)

Let u_{NN} be a scalar-valued FFNN and let us split its corresponding set of learnable parameters θ as follows:

$$\theta := (\eta, \mathbf{W}) \in \Theta = \Lambda \times \mathbb{R}^{n_{K+1}}, \tag{2.31a}$$

where

$$\eta := \{\mathbf{W}_j, \mathbf{b}_j\}_{j=1}^{K} \in \Lambda = \{\mathbb{R}^{n_j \times n_{j-1}} \times \mathbb{R}^{n_j}\}_{j=1}^{K}. \tag{2.31b}$$

Then, fixed some $\eta \in \Lambda$, we obtain a corresponding vector space $\mathbb{U}_\eta$ spanned by the component functions of $\mathbf{y}_K(\cdot\,;\eta)$,

$$\mathbb{U}_\eta := \mathrm{span}\big\{\mathbf{y}_K(\cdot\,;\eta)\big\} = \left\{ \sum_{i=1}^{n_K} w_i \; y_K^{(i)}(\cdot\,;\eta_i) : w_i \in \mathbb{R} \right\}. \tag{2.32a}$$

Here, $y_K^{(i)}(\cdot\,;\eta_i) : X \longrightarrow \mathbb{R}$ stands for the (scalar-valued) i^{th} component function of

$$\mathbf{y}_K(\cdot\,;\eta) = \begin{bmatrix} y_K^{(1)}(\cdot\,;\eta_1) \\ y_K^{(2)}(\cdot\,;\eta_2) \\ \vdots \\ y_K^{(n_K)}(\cdot\,;\eta_{n_K}) \end{bmatrix} : X \longrightarrow \mathbb{R}^{n_K}, \tag{2.32b}$$

with η_i denoting the reduced version of $\eta = \{\mathbf{W}_j, \mathbf{b}_j\}_{j=1}^{K}$ that in $\mathbf{W}_K$ and in $\mathbf{b}_K$ excludes the entries that are independent of the production of $y_K^{(i)}$, namely,

$$\eta_i = \{\mathbf{W}_j, \mathbf{b}_j\}_{j=1}^{K-1} \cup \{W_K^{(i,r)}\}_{r=1}^{n_{K-1}} \cup \{b_K^{(i)}\}, \tag{2.32c}$$

with $W_j^{(i,r)}$ and $b_j^{(i)}$ being the $(i,r)^{\text{th}}$ and i^{th} entries of $\mathbf{W}_K$ and $\mathbf{b}_K$, respectively.

Hence,

$$\mathbb{U}_{\text{NN}} = \bigcup_{\eta \in \Lambda} \mathbb{U}_\eta. \tag{2.33}$$

Vector-valued FFNN ($n_{K+1} > 1$)

All of the above is scalable to a vector-valued case by constructing the Cartesian product vector space $\mathbb{U}_\eta^{n_{K+1}}$ consisting of n_{K+1} copies of $\mathbb{U}_\eta$, i.e.,

$$\mathbb{U}_\eta^{n_{K+1}} = \mathbb{U}_\eta \times \mathbb{U}_\eta \times \cdots \times \mathbb{U}_\eta, \qquad (n_{K+1} \text{ times}). \tag{2.34}$$

Then,

$$\mathbb{U}_{\text{NN}} = \bigcup_{\eta \in \Lambda} \mathbb{U}_\eta^{n_{K+1}}, \tag{2.35a}$$

where each element in $\mathbb{U}_\eta^{n_{K+1}}$ is of the form

$$\mathbf{u}_{\text{NN}}(\cdot\,;\eta,\mathbf{W}) = \begin{bmatrix} \sum_{j=1}^{n_K} W^{(1,r)}\; y_K^{(r)}(\cdot\,;\eta_r) \\ \sum_{j=1}^{n_K} W^{(2,r)}\; y_K^{(r)}(\cdot\,;\eta_r) \\ \vdots \\ \sum_{j=1}^{n_K} W^{(n_{K+1},r)}\; y_K^{(r)}(\cdot\,;\eta_r) \end{bmatrix} \in \mathbb{U}_\eta^{n_{K+1}} = \begin{bmatrix} \mathbb{U}_\eta \\ \mathbb{U}_\eta \\ \vdots \\ \mathbb{U}_\eta \end{bmatrix}. \tag{2.35b}$$

By construction, for any $\eta \in \Lambda$,

$$\dim(\mathbb{U}_\eta) \le n_K, \qquad \dim(\mathbb{U}_\eta^{n_{K+1}}) = n_{K+1}\dim(\mathbb{U}_\eta) \le n_{K+1}n_K, \tag{2.36}$$

because the linear independence among the component functions of $\mathbf{y}_K(\cdot\,;\eta)$ is generally not guaranteed.

3. Deep elementu finituen metodoa*

Laburpena. Deribatu partzialetako ekuazio parametriko linealak ebazteko ikasketa sakoneko arkitektura dinamiko bat aurkezten dugu, elementu finituen metodoan oinarrituta dagoena. Arkitekturako neuronen arteko loturek elementu finituetako grafoetakoak antzeratzen dituzte, horietan diskretizazio-sarea fintzen denean. Lan honetan, zenbait galera-funtzio aztertu ditugu, konbergentzia hobetzeko hainbat aurrebaldintzatzaile eta norma erabiliz. Sinpletasunagatik, metodoaren inplementazioa dimentsio bakarreko (1Dko) espazioko eremuan gauzatu dugu, nahiz eta 2D eta 3D problemetara ere heda daitekeen. Egin ditugun zenbakizko esperimentuek, orokorrean, agerian uzten dute metodoak balio duela problema simetriko positibo definituetarako edo problema zehaztugabeetarako, bai ekuazio parametrikoetan baita ez-parametrikoetan ere. Hala ere, kasu batzuetan, ganbiltasun ezak eragotzi egiten du zehaztasun handiko soluzioak lortzea. *Argitaratutako bertsiorako, ikus [218].*

3.1. Sarrera

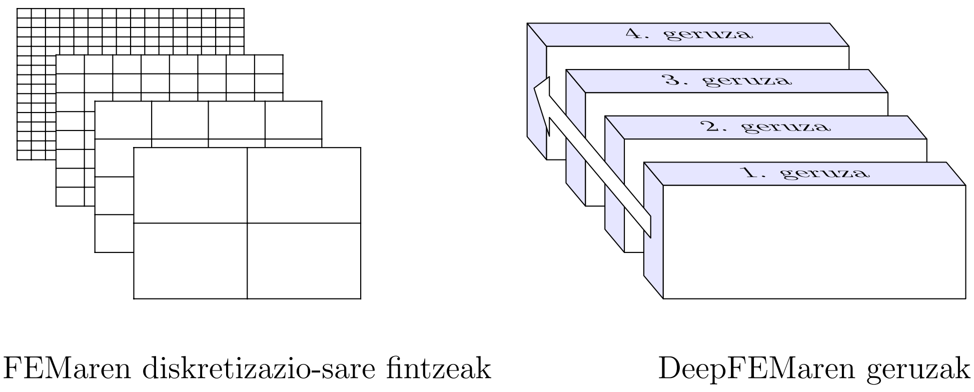
Berrikuspen literarioko lan ugaritan lantzen da PDE (parametriko) baten soluzioa hurbilduko duen funtzio jarraitu bat (NNa) aurkitzeko ideia (gogoratu 1.4. atala). Oro har, diseinu horiek NNa eremuko edozein puntutan ebaluatzeko aukera ematen dute, hau da, diskretizazio-sarearen mendekotasunik ez duen egitura daukate. Hala ere, badituzte muga batzuk. Honako bi hauek nabarmenduko ditugu: (a) gehienetan, NN arkitekturek ez daukate azaltzeko gaitasunik [194, 21], eta (b) galera-funtzioaren barruan zenbakizko integrazio arauak diseinatzea ez da izaten erraza, [115]-en esaten den bezala, eta xehetasun handiagoz [186]-n.

Aipatutako bi zailtasunak gainditzeko, PDE parametrikoen ebazpenerako FEM-ean oinarritutako ikaskuntza sakoneko metodo bat proposatzen dugu, *Deep elementu finituen metodoa (DeepFEM)*[2] edo *elementu finituen metodo sakona* deitua. Proposatutako NNaren arkitekturak FEMa aplikatu ostean lortzen den ekuazio linealetako sistema parametrikoaren ebazle gisa jardungo du, diskretizazio-

*See Appendix B for the English version/Ikus B. eranskina ingelesezko bertsiorako.

[2]Ingelesetik, *Deep Finite Element Method (DeepFEM).*

sarea fintzean elementu finituen konektagarritasun-grafoa imitatuz. NNaren geruza bakoitzak ResNet diseinua du [93] eta diskretizazio-sare zakarragoetako soluzioak diskretizazio-sare finagoetara hedatzen ditu. 3.1. irudiak FEMeko diskretizazio-sareen fintzeen eta NNko geruzen arteko erlazioa erakusten du. Horrela, arkitekturak neurri bateko azalgarritasuna du, eta sarearen iragarpena diskretizazio-sare fineko nodoetako balioez osatutako bektorea da. Gainera, ikuspegi diskretu horrek zenbakizko integrazio zehatza ahalbidetzen du entrenamenduan zehar, NNak iragartzen duen soluzioa zatika definitutako polinomioen espazioan dagoelako.



3.1. irudia: FEMaren diskretizazio-sare fintzeen eta DeepFEMaren geruzen arteko lotura.

DeepFEMak, lehendabizi, hasierako arkitektura bat ezartzen du. Arkitektura hori, entrenamenduaren ostean, gai izango da soluzio zakarrak iragartzeko. Gero, prozesu iteratibo eta dinamiko bat jarraituz, arkitekturan geruzak sartuz joango gara, aurretik entrenatutako aldagaiak[3] mantenduz eta berriak gehituz. Ondoren, eredu berriaren aldagaiak berriro entrenatuko ditugu; eta prozesu bera errepikatuko da nahi dugun zehaztasuneko soluzioa lortu arte. Horrela, proposatutako NNak aukera ematen digu diskretizazio-sarearen konbergentzi-azterketa egiteko.

Gure inplementazioa *dimentsio bakarreko (1Dko)*[4] problemetara mugatzen da, zatika linealak diren hurbilpenen bidez diskretizazio-sare uniformeen gainean. Hala ere, lana zuzenean heda daiteke dimentsio handiagoko problemetara, diskretizazio-sare handiagoko hurbilketa polinomikoetara, edota egokitutako diskretizazio-sareetara. Hala ere, inplementazio landuagoa behar da irizpide geometrikoak kontuan izateko eta diskretizazio-sarearen fintzearen eraginez nodoen zenbaketa

[3]Kapitulu honetan zehar, 2. kapituluan erabilitako *parametro entrenagarrien* terminologia *aldagai entrenagarrietara* aldatuko dugu, PDEaren koefizienteak eta NNaren pisuak eta alborapenak (ingelesetik, *bias terms*) bereizteko.

[4]Ingelesetik, *one-dimensional (1D)*.

egiteko. Lan honetan ez dugu gai horietan sakonduko. Hainbat eredu problematan lortu ditugun zenbakizko emaitzak erakutsiko ditugu, bai parametro konstanteak erabili direnean baita zatika kostanteak erabili direnean ere.

Aurkeztutako teknologiaren ekarpen nagusia da NNak problema parametrikoak ebazteko duen gaitasuna agerian uztea. Adibide gisa, lehenengo, parametrikoak ez diren kasuak aurkeztuko ditugu, eta, ondoren, parametrikoak. Gauza bera egingo dugu zenbakizko emaitzak deskribatzean; metodoak ingurune ez-parametrikoetan duen ulergarritasunak eta mugak kasu parametrikora heda baitaitezke.

Kapitulu honen gainerako zatiak honako hauek dira. 3.2. atalean gure intereseko problema eta hari dagokion formulazio bariazionala azalduko dira. 3.3. atalak DeepFEM arkitektura deskribatuko du eta 3.4. atalak hautatutako galera-funtzioa definituko du. 3.5. atalak TF liburutegia [3, 2] erabiltzean aurkitu ditugun inplementazio-mugak eta xehetasunak erakutsiko ditu. Amaitzeko, 3.6. atalean lortutako zenbakizko emaitzak eztabaidatuko ditugu.

3.2. Eredu problema eta elementu finituen formulazioa

BVP parametriko jakin bat hartuko dugu, nahiz eta aurkeztuko dugun metodologia FEMa erabiliz ebatz daitezkeen beste problema batzuetarako ere aplikagarria izan.

Izan bedi Ω eremu leun bat eta ezar dezagun honako BVP parametrikoa:

$$\begin{cases} -\nabla \cdot \sigma \nabla u + \alpha u = f, & \Omega\text{-n}, \\ u = 0, & \Gamma_D\text{-n}, \\ -\sigma \frac{\partial u}{\partial n} = g, & \Gamma_N\text{-n}, \end{cases} \tag{3.1}$$

non $\sigma > 0$ eta $\alpha \in \mathbb{R}$ parametroak zatika konstanteak diren funtzioak baitira, f iturria, eta g Neumannen datua. Γ_D eta Γ_N Dirichlet eta Neumann mugaldeak dira, hurrenez hurren. n kanporanzko bektore normala da Γ_N-ren puntu bakoitzean, eta $\partial u / \partial n := \nabla u \cdot n$. Kontuan izan difusio-erreakzio eredu orokor honek Poissonen ($\alpha = 0$) eta Helmholtzen ($\alpha < 0; \sigma = 1$) ekuazioak barne hartzen dituela.

Goiko BVPren formulazio bariazional bat honako hau da:

$$\begin{cases} \text{Aurkitu } u^* \in H_0^1(\Omega) \text{ izanik, honako hau beteko duena:} \\ (\sigma \nabla u^*, \nabla v)_\Omega + (\alpha u^*, v)_\Omega = (f, v)_\Omega - (g, v)_{\Gamma_N}, \forall v \in H_0^1(\Omega), \end{cases} \tag{3.2}$$

non

$$(u, v)_\Omega := \int_\Omega u \cdot v, \tag{3.3}$$

eta hurrengo espazioak kontuan hartuta:

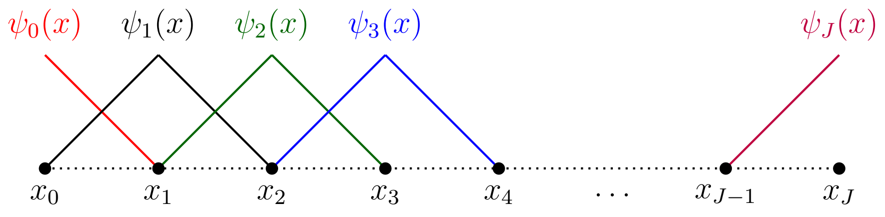
$$L^2(\Omega) = \{u : \Omega \longrightarrow \mathbb{R} : (u,u)_\Omega < \infty\}, \tag{3.4a}$$
$$H^1(\Omega) = \{u \in L^2(\Omega) : (\nabla u, \nabla u)_\Omega < \infty\}, \tag{3.4b}$$
$$H_0^1(\Omega) = \{u \in H^1(\Omega) : u|_{\Gamma_D} = 0\}. \tag{3.4c}$$

Prozesu osoan zehar suposatuko dugu f eta g nahiko erregularrak direla, hau da, $f \in L^2(\Omega)$ eta $g \in H^{1/2}(\partial\Omega)$. Sobolev espazio mugatuei/zatikariei buruzko eztabaida zehatz baterako, ikus [74].

1Dko FEM formulazioa erabilita, honako forma hau daukan soluzioa aurkitu nahi dugu:

$$u_{\text{FEM}}(x;\sigma,\alpha) := \sum_{j=0}^{J} u_{\text{FEM},j}(\sigma,\alpha)\ \psi_j(x), \tag{3.5}$$

non $u_{\text{FEM},j}(\sigma,\alpha)$ koefiziente ezezagunak izanik eta $\psi_j(x)$ zatika linealak diren denda itxuradun oinarriko funtzioak (ikus 3.2. irudia).



3.2. irudia: Denda itxuradun eta zatika linealak diren oinarriko funtzioen, ψ_j-en, euskarria.

(3.5) eta (3.2) ekuazioak konbinatuz eta ψ_j oinarriko funtzioekin testatuz, honako ekuazio linealetako sistema honetara iritsiko gara:

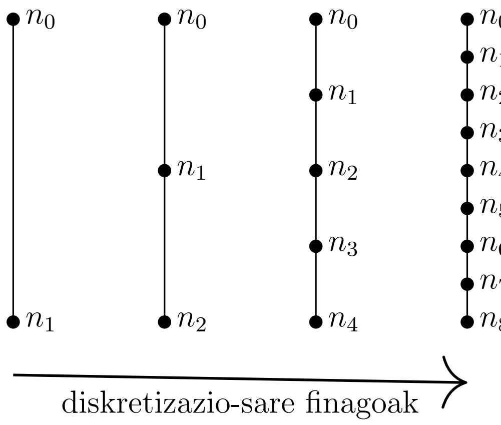
$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{3.6}$$

non $\mathbf{A} = \mathbf{A}(\sigma,\alpha) := [(\sigma\psi_j', \psi_i')_\Omega + (\alpha\psi_j, \psi_i)_\Omega]_{i,j=0}^J$, $\mathbf{u} := \mathbf{u}_{\text{FEM}}(\sigma,\alpha) := [u_{\text{FEM},j}]_{j=0}^J$ koefiziente ezezagunen bektorea izanik, eta $\mathbf{f} := [(f,\psi_j)_\Omega - (g,\psi_j)_{\Gamma_N}]_{j=0}^J$ karga bektorea baita.

3.3. Arkitektura dinamikoa

Lehenengo, problema ez-parametrikoetarako deskribatuko dugu proposaturiko arkitektura, eta, gero, kasu parametrikora hedatuko dugu. Azkenik, parametroak konstanteak edota zatika konstanteak direneko kasuak aztertuko ditugu.

3.3. irudiak erakusten du fintze uniformeak egiterakoan nodoak zenbatzeko aukeratu dugun modua. Horren arabera, $\mathbf{E}$ zabaltze-operadorea matrize sakabanatu baten bidez emana dator. Matrize hori 1 edo 1/2 zenbakiez beteta dago, nodo bakoitza diskretizazio-sare zakarragotik finagora hedatzen den moduaren arabera. Kontuan izan, antzeko zabaltze-operadoreak existitzen direla *bi dimentsioko (2Dko)* [5] eta *hiru dimentsioko (3Dko)*[6] problemetan, edota ordena handiagoko elementuetan, baita $H(\mathrm{div})$, $H(\mathrm{curl})$ eta L^2 diskretizazioetan ere [57].



3.3. irudia: 1Dko diskretizazio-sare baten fintze uniformeen eta nodoen zenbaketaren progresioa.

3.3.1. Eskema ez-parametrikoa koefiziente konstantedunetarako

Izan bitez σ and α konstante errealak. Elementu bakar bateko diskretizazio-sarerako, honako bi-geruzako sakonera duen arkitektura hartuko dugu:

$$\mathbf{u}_{\mathrm{NN}}^{(1)}(\sigma,\alpha) = \mathbf{W}^{(1)}\ \varphi\left(\mathbf{W}_{\sigma,\alpha}^{(1)} \begin{bmatrix}\sigma \\ \alpha\end{bmatrix} + \mathbf{b}_{\sigma,\alpha}^{(1)}\right) \in \mathbb{R}^2, \tag{3.7}$$

non φ aktibazio funtzioa baita eta output bektorea bi dimentsiokoa izanik, (elementu bakarreko) diskretizazio-sare zakarrean FEMaren soluzioa hurbiltzea helburu duena ($s = 1$ urratsa), hau da,

$$\mathbf{u}_{\mathrm{FEM}}^{(1)}(\sigma,\alpha) = \begin{bmatrix} u_{\mathrm{FEM},0}^{(1)}(\sigma,\alpha) \\ u_{\mathrm{FEM},1}^{(1)}(\sigma,\alpha)\end{bmatrix} \approx \begin{bmatrix} u_{\mathrm{NN},0}^{(1)}(\sigma,\alpha) \\ u_{\mathrm{NN},1}^{(1)}(\sigma,\alpha)\end{bmatrix} = \mathbf{u}_{\mathrm{NN}}^{(1)}(\sigma,\alpha). \tag{3.8}$$

[5]Ingelesetik, *two-dimensional (2D)*.
[6]Ingelesetik, *three-dimensional (3D)*.

Entrenagarriak diren aldagaien multzoa honako hau da:

$$\theta^{(1)} = \{\mathbf{W}^{(1)}_{\sigma,\alpha}, \mathbf{b}^{(1)}_{\sigma,\alpha}, \mathbf{W}^{(1)}\}. \tag{3.9}$$

Gero, diskretizazio-sarea finduko dugu bi elementuko diskretizazio-sarea lortu arte ($s = 2$ urratsa). Orduan, sarrera-mendekotasuna daukan ResNet [93] motako geruza bat sartuko dugu $\mathbf{u}^{(1)}_{\text{NN}}$ arkitekturan $\mathbf{u}^{(2)}_{\text{NN}}$ lortzeko:

$$\mathbf{r}^{(2)}_{\text{NN}}(\sigma,\alpha) = \mathbf{W}^{(2)}\,\varphi\left(\mathbf{W}^{(2)}_{\sigma,\alpha}\begin{bmatrix}\sigma\\ \alpha\end{bmatrix} + \mathbf{b}^{(2)}_{\sigma,\alpha}\right) \in \mathbb{R}^3, \tag{3.10a}$$

$$\mathbf{u}^{(2)}_{\text{NN}}(\sigma,\alpha) = \mathbf{E}^2_1\mathbf{u}^{(1)}_{\text{NN}}(\sigma,\alpha) + \mathbf{r}^{(2)}_{\text{NN}}(\sigma,\alpha) \in \mathbb{R}^3, \tag{3.10b}$$

non $\mathbf{E}^2_1$ zabaltze-matrizea baita, $\mathbf{u}^{(1)}_{\text{NN}}$ bektorea diskretizazio-sare finean (bi elementukoan) egon dadin. Orain, aldagai entrenagarrien $\theta^{(2)}$ multzoa honako hau da:

$$\{\mathbf{W}^{(2)}_{\sigma,\alpha}, \mathbf{b}^{(2)}_{\sigma,\alpha}, \mathbf{W}^{(2)}\} \qquad \text{edo} \qquad \theta^{(1)} \cup \{\mathbf{W}^{(2)}_{\sigma,\alpha}, \mathbf{b}^{(2)}_{\sigma,\alpha}, \mathbf{W}^{(2)}\} \tag{3.11}$$

entrenamendua *geruzaz geruza* edo *hasieratik amaieraraino* nola egiten dugun kontuan hartuz, hurrenez hurren. Kontuan izan aldagai berriak $\mathbf{r}^{(2)}_{\text{NN}}$-ri bakarrik dagozkiola. Berrentrenamenduaren hasieran $\mathbf{r}^{(2)}_{\text{NN}} = 0$ izan dadin aldagai berriak hasieratuko ditugu (adibidez, $\mathbf{W}^{(2)}, \mathbf{b}^{(2)} = 0$ hasieratuz), eta aurreko urratsean ikasitako aldagaiak, $\theta^{(1)}$, mantenduko ditugu $s = 2$ urratsaren hasieran. $\mathbf{u}^{(2)}_{\text{NN}}$ berriro entrenatuko dugu (geruzaz geruza edo hasieratik amaierarainoko entrenamenduekin) FEMaren soluzioa hurbil dezan (bi elementuko) diskretizazio-sare finean, hau da,

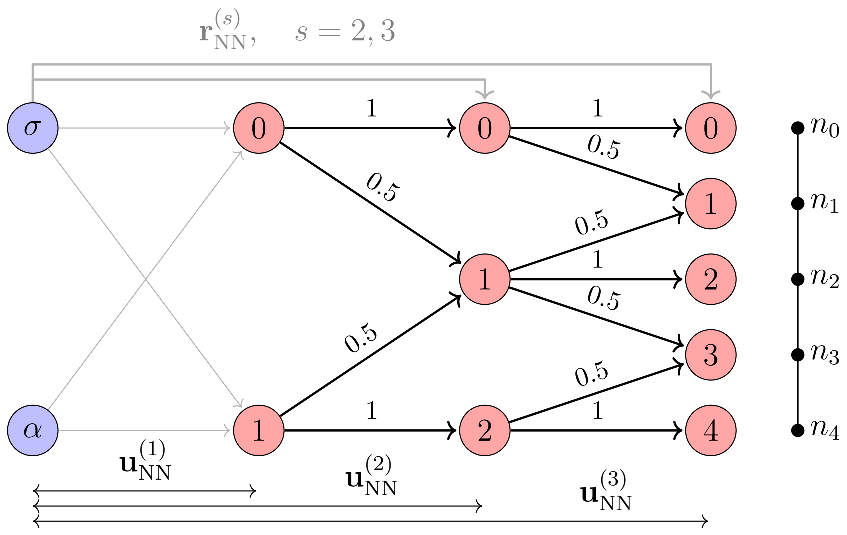
$$\mathbf{u}^{(2)}_{\text{FEM}}(\sigma,\alpha) = \begin{bmatrix} u^{(2)}_{\text{FEM},0}(\sigma,\alpha)\\ u^{(2)}_{\text{FEM},1}(\sigma,\alpha)\\ u^{(2)}_{\text{FEM},2}(\sigma,\alpha)\end{bmatrix} \approx \begin{bmatrix} u^{(2)}_{\text{NN},0}(\sigma,\alpha)\\ u^{(2)}_{\text{NN},1}(\sigma,\alpha)\\ u^{(2)}_{\text{NN},2}(\sigma,\alpha)\end{bmatrix} = \mathbf{u}^{(2)}_{\text{NN}}(\sigma,\alpha). \tag{3.12}$$

Prozesu hori urratsez urrats errepikatuko dugu, NNaren sakontasuna handituz, diskretizazio-sarearen elementu kopurua soluzio analitikoa zehazki hurbiltzeko gai den arte. 3.4. irudian ikus daiteke arkitektura dinamiko hori.

3.3.2. Eskema parametrikoa koefiziente konstantedunetarako

Eskema ez-parametrikoa dimentsio oso txikiko pisuak eta alborapenak ezartzean zetzan (adibidez, $\mathbf{W}^{(1)}, \mathbf{W}^{(1)}_{\sigma,\alpha} \in \mathbb{R}^{2\times 2}$ eta $\mathbf{W}^{(2)} \in \mathbb{R}^{3\times 2}, \mathbf{W}^{(2)}_{\sigma,\alpha} \in \mathbb{R}^{2\times 2}$), koefizienteen lagin bakar batean entrenatu nahi baikenuen.

Problema parametrikoetarako (koefizienteen laginen datu-base batean entrenatzeko), sakonera eta zabalera gehituko dizkiegu arkitektura dinamikoaren atal


$\mathbf{r}_{\mathrm{NN}}^{(s)}, \quad s = 2, 3$
σ
α
0
1
0
1
2
0
1
2
3
4
1
0.5
n_0
n_1
n_2
n_3
n_4
$\mathbf{u}_{\mathrm{NN}}^{(1)}$
$\mathbf{u}_{\mathrm{NN}}^{(2)}$
$\mathbf{u}_{\mathrm{NN}}^{(3)}$

3.4. irudia: Eskema ez-parametrikoaren DeepFEM arkitektura dinamikoaren irudikapena koefiziente konstantedunetarako. Gezi grisek aldagai entrenagarriak adierazten dituzte, eta gezi beltzek, berriz, (entrenaezinak diren) zabaltze-operazioak.

entrenagarriei, honela:

$$\mathbf{u}_{\mathrm{NN}}^{(1)} = \mathcal{FC}_{\mathrm{NN}}^{(1)}(\sigma, \alpha), \tag{3.13a}$$

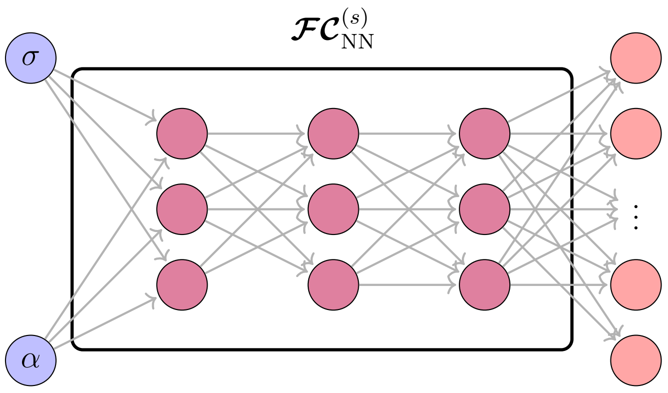
$$\mathbf{r}_{\mathrm{NN}}^{(s)} = \mathcal{FC}_{\mathrm{NN}}^{(s)}(\sigma, \alpha), \qquad s \geq 2, \tag{3.13b}$$

$$\mathbf{u}_{\mathrm{NN}}^{(s)} = \mathbf{E}_{s-1}^{(s)} \mathbf{u}_{\mathrm{NN}}^{(s-1)} + \mathbf{r}_{\mathrm{NN}}^{(s)}, \qquad s \geq 2. \tag{3.13c}$$

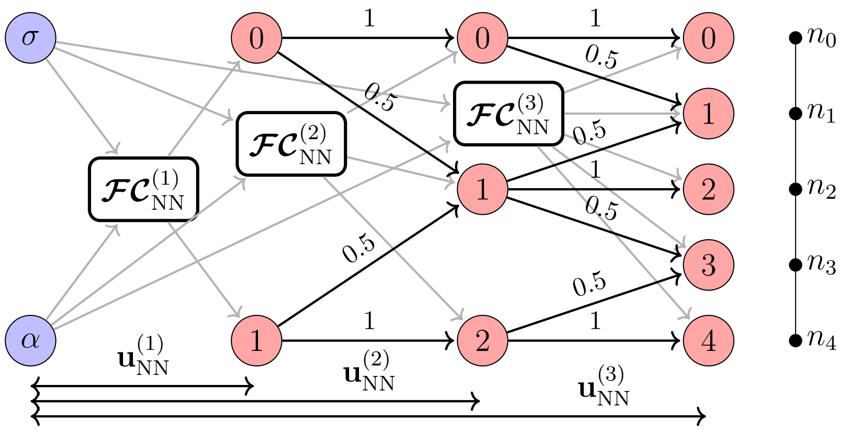
Hemen, $\mathcal{FC}_{\mathrm{NN}}^{(s)}$-k adierazten du aktibatu gabeko azken geruza batekin guztiz konektatutako FFNN bat (gogoratu 2.1. atala), eta horren irteera-dimentsioa zehaztuko da FEM diskretizazioaren dimentsioekin bat etortzeko eran. Sinpletasunagatik, *bloke entrenagarri* deituko diogu $\mathcal{FC}_{\mathrm{NN}}^{(s)}$ ereduei. 3.5. irudiak bloke entrenagarriaren arkitektura erakusten du eta 3.6. irudiak horiek irudikatzen ditu DeepFEM eskema parametrikoaren barruan.

3.3.3. Eskema parametrikoa zatika konstanteak diren koefizienteak dituztenetarako

Azkenik, zatika konstanteak diren koefizienteen portaera erantsiko diegu parametroei. Horretarako, pentsatuko dugu parametroek balio konstanteak dituztela hasierako diskretizazio-sareko elementu bakoitzean. Horrela, arkitekturaren aldaketek soilik sarrerako geruzan dute eragina: σ eta α balio errealen ordez, hasierako

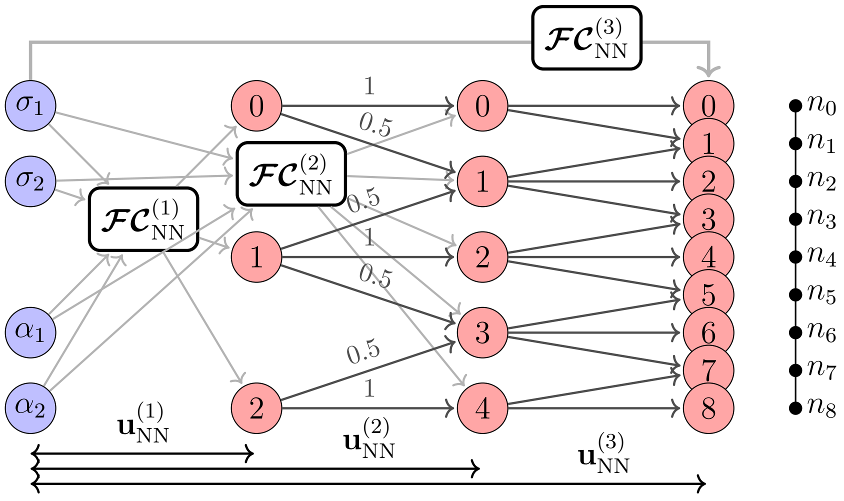


3.5. irudia: *s*. urratsean bloke entrenagarri baten arkitekturaren irudikapena. Gezi grisek aldagai entrenagarriak adierazten dituzte.


σ
α
$\mathcal{FC}_{\text{NN}}^{(1)}$
$\mathcal{FC}_{\text{NN}}^{(2)}$
$\mathcal{FC}_{\text{NN}}^{(3)}$
0
1
0
1
2
0
1
2
3
4
1
0.5
0.5
1
0.5
0.5
1
0.5
1
0.5
1
n_0
n_1
n_2
n_3
n_4
$\mathbf{u}_{\text{NN}}^{(1)}$
$\mathbf{u}_{\text{NN}}^{(2)}$
$\mathbf{u}_{\text{NN}}^{(3)}$

3.6. irudia: Eskema parametrikoaren DeepFEM arkitektura dinamikoaren irudikapena koefiziente konstantedun problemarako. Gezi grisek aldagai entrenagarriak adierazten dituzte, eta gezi beltzek, berriz, (entrenaezinak diren) zabaltze-operazioak.

diskretizazio-sarearen elementu kopuruarekin bat datorren dimentsioko bektoreak erabiliko dira. 3.7. irudiak arkitekturaren grafo dinamiko eta parametrikoa erakusten du, bi elementuko hasierako diskretizazio-sarearentzat.



$\mathcal{FC}_{\mathrm{NN}}^{(3)}$
$\mathcal{FC}_{\mathrm{NN}}^{(2)}$
$\mathcal{FC}_{\mathrm{NN}}^{(1)}$
σ_1
σ_2
α_1
α_2
1
0.5
0.5
1
0.5
0.5
1
n_0
n_1
n_2
n_3
n_4
n_5
n_6
n_7
n_8
$\mathbf{u}_{\mathrm{NN}}^{(1)}$
$\mathbf{u}_{\mathrm{NN}}^{(2)}$
$\mathbf{u}_{\mathrm{NN}}^{(3)}$

3.7. irudia: Eskema parametrikoaren DeepFEM arkitektura dinamikoaren irudikapena zatika konstanteak diren koefizienteak dituztenetarako. Gezi grisek aldagai entrenagarriak adierazten dituzte, eta gezi beltzek, berriz, (entrenaezinak diren) zabaltze-operazioak.

3.4. Galera-funtzioa eta entrenamendua

3.3. atalean urratsez urrats deskribatutako metodologiaren barruan, “(s)” goi-indizea ezabatuko dugu sinpletasunagatik, hautatutako aldagai entrenagarriak, galera-funtzioak eta normak urratsaren mende daudela ulertuta.

NNak FEMa aplikatu ostean sortzen den ekuazio linealetako sistema parametrikoaren soluzioa hurbil dezan, hau da,

$$\mathbf{u}_{\mathrm{NN}} \approx \mathbf{u}_{\mathrm{FEM}}, \tag{3.14}$$

honako galera-funtzioa aukeratuko dugu:

$$\mathcal{L}(\theta, \{\sigma_i, \alpha_i\}_{i=1}^{N}) := \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{A}(\sigma_i, \alpha_i)\mathbf{u}_{\mathrm{NN}}(\sigma_i, \alpha_i; \theta) - \mathbf{f}\|, \tag{3.15}$$

non $\mathbf{u}_{\mathrm{NN}}(\sigma_i, \alpha_i; \theta)$ DeepFEM ereduaren iragarpena baita i. laginerako, $\mathbf{A}(\sigma_i, \alpha_i)$ FEM matrizea —gogoratu (3.6)—, $\mathbf{A}(\sigma_i, \alpha_i)\mathbf{u}_{\mathrm{NN}}(\sigma_i, \alpha_i; \theta) - \mathbf{f}$ hondar bektorea, eta $\|\cdot\|$ ezarritako norma bektorial bat (adibidez, 2-norma).

3.4.1. Gradientean oinarritutako entrenamendua

Galera-funtzioa minimizatzeko, Adam optimizatzailea [119] aplikatuko dugu urrats bakoitzaren hasieran. Exekuzioa gelditukoa dugu iterazio kopurua gehienezko balio batera iristen denean. Gainera, ez atzera ez aurrera gera ez dadin, iterazio bakoitzeko galera zenbait iterazio aurrerago lortutako galerarik txikienarekin konparatuko dugu. Zenbait iterazioren ostean, galeraren hobekuntza hutsala bada, exekuzioa gelditu egingo dugu. Adam optimizatzailea darabilgun bitartean galera areagotu edo gutxitu egin daitekeenez, exekuzioan zehar galera hori ikuskatuko dugu, eta prozesuaren amaieran, aldagaien konfiguraziorik onena itzuliko dugu emaitza gisa. Ondoren, ohiko gradientearen jaitsieran oinarritutako optimizatzaile bat erabiliko dugu, galeraren araberako ikaskuntza-tasa adaptatibo bat duena eta lan honetarako guk diseinatu duguna, *Adalr* izenekoa. Adam-en kasuan erabilitako irizpide berberak erabiliko ditugu Adalr-en exekuzioa gelditzeko ere. Bestalde, Adalr-ek aldagaiak egokitu egingo ditu galeraren areagotzea gerta ez dadin. 1. algoritmoak erakusten du ikaskuntza-tasaren adaptibitatearen portaera eta aldagaiak onartzeko edo baztertzeko irizpidea. Zenbakizko emaitzetan bi optimizatzaileak alderatuko ditugu. Ikuspegi orokor gisa, Adam-ek hasieran galera azkar gutxitzen duen arren, optimoaren azpitik geratzen den atalase bateraino egingo du, eta portaera oszilakorra erakutsiko du. Beste aldetik, Adalr-ek galera monotonikoki gutxituko du, Adam-ena baino konbergentzia abiadura txikiagoarekin.

3.4.2. Normaren hautaketa eta aurrebaldintzaketa

Norma diskretu bat aukeratu nahi dugu galera-funtzioan —gogoratu (3.15)— errorearen *energia-normaren* antzeko portaera duena. Simetrikoa eta positibo-definitua den problema batean, energia-norma honako era honetan emana dator:

$$\|\mathbf{v}\|_{\mathbf{A}} := \sqrt{\mathbf{v}^T\mathbf{A}\mathbf{v}}, \tag{3.16}$$

non $\mathbf{v}^T$ bektorea $\mathbf{v}$ zutabe bektorearen iraulia baita eta $\mathbf{A}$ ekuazio linealen sistemaren matrizea, simetrikoa eta positibo-definitua den matrizea. Errore bektorea horrela idazten badugu (σ, α) lagin bakoitzerako:

$$\mathbf{e}(\sigma,\alpha) := \mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{A}^{-1}(\sigma,\alpha)\mathbf{f} = \mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{u}_{\text{FEM}}(\sigma,\alpha), \tag{3.17}$$

hondarrari dagokion honako erlazio honetara iritsiko gara:

$$\|\mathbf{e}(\sigma,\alpha)\|_{\mathbf{A}(\sigma,\alpha)} = \|\mathbf{A}(\sigma,\alpha)\mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{f}\|_{\mathbf{A}^{-1}(\sigma,\alpha)}. \tag{3.18}$$

Praktikan, $\mathbf{A}(\sigma,\alpha)$-ren alderantzizkoa konputaezina da, eta, beraz, errorea ezezaguna da. Horregatik, ekuazio linealen sistema bat ebazteko metodo iteratiboetan gertatzen den bezala, alderantzizko operadorea antzeratzeko aurrebaldintzatzaile bat aukeratuko dugu, era horretan sistemaren baldintza-zenbakia gutxituz [169]. Orduan, honako norma bektoriala $\|\cdot\|$ definitzen dugu (3.15)-en:

$$\|\mathbf{v}\| := \|\mathbf{v}\|_{\mathbf{P}} = \sqrt{\mathbf{v}^T\mathbf{P}\mathbf{v}}, \tag{3.19}$$

1. algoritmoa: Adalr optimizatzailea

```
/* Goi-indizeek iterazioak adierazten dituzte                     */
Input: $\theta^{(0)}, \eta^{(0)}, D$ ;        // Hasierako aldagaiak eta datu basea
Output: $\theta^*$, $\mathcal{L}^*$ ;              // Amaierako aldagaiak eta galera
$\mathcal{L}^{(0)} = \mathcal{L}(\theta^{(0)}; D)$; $\theta^{(1)} = \theta^{(0)} - \lambda^{(0)} \frac{\partial \mathcal{L}}{\partial \theta}(\theta^{(0)}; D)$;
$\theta^* = \theta^{(0)}$; $\mathcal{L}^* = \mathcal{L}^{(0)}$; $t = 1$;
while ez gelditu do
    $\mathcal{L}^{(t)} = \mathcal{L}(\theta^{(t)}; D)$; $\theta^{(t+1)} = \theta^{(t)} - \lambda^{(t)} \frac{\partial \mathcal{L}}{\partial \theta}(\theta^{(t)}; D)$ ;
    /* Galeraren balio berria okerragoa bada                      */
    if $\mathcal{L}^{(t)} > \mathcal{L}^*$ then
        $\lambda^{(t+1)} =$ Gutxitu($\lambda^{(t)}$); $\theta^{(t+1)} = \theta^*$;
    else
        $\theta^* = \theta^{(t)}$;
        /* Konbergentzia geldoa bada                               */
        if konbergentzia geldoa and aldagaien bazterketarik eza then
            $\lambda^{(t+1)} =$ Handitu($\lambda^{(t)}$);
        else
            $\lambda^{(t+1)} = \lambda^{(t)}$;
        $\mathcal{L}^* = \mathcal{L}^{(t)}$;
    $t = t + 1$;
return $\theta^*$, $\mathcal{L}^*$
```

non $\mathbf{P} = \mathbf{P}(\sigma, \alpha)$ aurrebaldintzatzaile bat baita $\mathbf{A}(\sigma, \alpha)$ matrizearentzako. Tamaina ezberdineko blokekako Jacobi aurrebaldintzatzaileak aukeratuko ditugu, non elementu bat gainjarrita baitute [169]. Ohartu, $\mathbf{P} = \mathbf{I}$ identitatea denean, 2-norma daukagula ($\|\cdot\|_{\mathbf{I}} = \|\cdot\|_2$).

Problema zehaztugabea bada, positiboki definitutako operadore bat aukeratuko dugu, $\mathbf{B}$ matrizea dagokiona. Norma berrian honako hau da errorearen eta hondarraren arteko erlazioa:

$$\|\mathbf{e}(\sigma, \alpha)\|_{\mathbf{B}} = \|\mathbf{A}^{-1}(\sigma, \alpha)\left\{\mathbf{A}(\sigma, \alpha)\mathbf{u}_{\text{NN}}(\sigma, \alpha) - \mathbf{f}\right\}\|_{\mathbf{B}}. \tag{3.20}$$

Aurreko arrazoiketa bera jarraituz, (3.15)-i dagokion norma honako era honetan definituko dugu:

$$\|\mathbf{v}\| := \|\mathbf{P}\mathbf{v}\|_{\mathbf{B}} = \sqrt{\mathbf{v}^T\mathbf{P}^T\mathbf{B}\mathbf{P}\mathbf{v}} = \|\mathbf{v}\|_{\mathbf{P}^T\mathbf{B}\mathbf{P}}. \tag{3.21}$$

Berriz ere, $\mathbf{P} = \mathbf{B} = \mathbf{I}$ identitate operadoreak direnean, aurreko ekuazioa bat letorke 2-norma diskretuarekin.

Zenbakizko emaitzetan, problema positiboetarako energia-norma erabiltzeaz gain, ohikoa da L^2 eta H^1 normak erabiltzea, bai ikuskatzeko baita (aurrebaldintzatutako) hondarra eta errorea neurtzeko ere. Norma jarraitu horiek badituzte definizio diskretu baliokideak:

$$\|u\|_{L^2} = \sqrt{\mathbf{u}^T\mathbf{M}\mathbf{u}} = \|\mathbf{u}\|_{\mathbf{M}}, \tag{3.22}$$

$$\|u\|_{H^1} = \sqrt{\mathbf{u}^T\mathbf{M}\mathbf{u} + \mathbf{u}^T\mathbf{K}\mathbf{u}} = \|\mathbf{u}\|_{\mathbf{K}+\mathbf{M}}, \tag{3.23}$$

non $\mathbf{u} = [u_j]_{j=0}^J$ ebaluaketen bektorea baita diskretizazio-sareko nodoetan; ψ_j, j. nodoari lotutako oinarriko funtzioa baita; eta $\mathbf{M}$ eta $\mathbf{K}$ masa- eta zurruntasun-matrizeak, (r, s). matrizeen sarreretan $(\psi_s, \psi_r)_\Omega$-ren eta $(\psi_s', \psi_r')_\Omega$-ren bidez definutakoak, hurrenez hurren. Hala komeni delako, (3.22) eta (3.23) norma diskretuak haien bertsio jarraituetatik jasotako izenak erabilita izendatuko ditugu, hau da, L^2 eta H^1 normak, hurrenez hurren.

Era berean, letra lodia kenduko dugu dagozkien funtzio bektorialen funtzio eskalarrak aipatzeko. Zehazki, NN iragarpenetarako eta FEM soluzioetarako,

$$u_{\text{NN}}(x; \sigma, \alpha) = \sum_{j=0}^{J} u_{\text{NN},j}(\sigma, \alpha)\ \psi_j(x), \tag{3.24a}$$

$$u_{\text{FEM}}(x; \sigma, \alpha) = \sum_{j=0}^{J} u_{\text{FEM},j}(\sigma, \alpha)\ \psi_j(x), \tag{3.24b}$$

non $u_{\text{NN},j}(\sigma, \alpha)$ eta $u_{\text{FEM},j}(\sigma, \alpha)$ DeepFEM iragarpen and FEM soluzio bektoreen, $\mathbf{u}_{\text{NN}}(\sigma, \alpha)$ eta $\mathbf{u}_{\text{FEM}}(\sigma, \alpha)$, j. osagaiak baitira, hurrenez hurren.

3.5. Inplementazioa

Python programazio-lengoaian inplementatu dugu kodea, eta TensorFlow 2 (TF2) [3, 2], NumPy [91], eta SciPy [224] liburutegiak erabili ditugu NNko ereduak eraikitzeko eta parametroak eta FEMaren datuak sortzeko. Geruzak eraikiko ditugu Keras-en; zehazki, TF2-aren barruan[7] dauden oinarriko klaseak berdefinituz (`tf.keras`). TF2-aren barruko liburutegi zehatz bat erabili dugu (`tf.sparse`) zabaltze-operadoreak eta FEM matrizeak kudeatzeko, memoria murriztuz eta errendimendu handiagoa lortuz. Zehaztasun bikoitza (float64) erabiliko dugu, zehaztasun sinplearen ordez (float32). Galera kalkulatuko dugu entrenagarria ez den geruza batean, $\mathbf{u}_{\mathrm{NN}}$ eredu nagusia aplikatu osteko urrats bakoitzean. Beherago azalduko ditugu inplementazioaren arloan aurkitu ditugun hiru zailtasun nagusiak.

3.5.1. Sakabanatutako tentsoreen multzokaketaren birmoldaketa

Keras ereduetako tentsoreak laginaren multzokaketaren lehen dimentsioa (ardatza ere esaten zaio) mantentzeko eran diseinatuta daude. Ondorioz, galeran egiten diren matrizeen eta bektoreen arteko biderketak 3Dko tentsore baten (sakabanatutako matrizeen multzokaketa) eta 2Dko tentsore baten (bektoreen multzokaketa) arteko eragiketa bihurtzen dira Keras ereduan. Bi tentsoreak trinkoak izango balira, aipatutako eragiketa Einstein batuketaren bidez definituko genuke. Hala ere, oraindik ez dago TF2-an funtzio baliokiderik sakabanatutako tentsoreak kudeatzeko[8]. Hori konpontzeko 3Dko tentsore sakabanatua berdimentsionatuko dugu blokeka gainjartzen ez diren 2Dko matrize batean jarriz. Era berean, 2Dko tentsorea 1Dko bektore luze batean jarriko dugu. Horren ondorioz, multzokaketari lotutako jarioa galdu egingo da, eta Keras-en lehenetsita dagoen entrenamendurako funtzioa ez erabiltzea eragingo du horrek. Horrela, ADaren bidezko maila baxuko optimizazioa egingo dugu *exekuzio sutsuan* (ingelesetik, *eager execution*), *grafo exekuzioan* (ingelesetik, *graph execution*) baino askoz motelagoa dena[9].

[7]TF1.X-en garaian, Keras kanpo-liburutegi bat zen TF-rako Pythonen erabilera errazteko. 2019ko irailean, TF2 irten zenetik, Keras TF2-ren azpiliburutegi bihurtu zen TF2.12 bertsioa 2023ko martxoan irten zenera arte, non Keras (berriro ere) TF-ren gainean eraikitako, eta JAX eta PyTorch sostengatzen dituen kanpoko esparru bihurtu baitzen. Ikus `https://keras.io` xehetasun gehiagorako (azken sarrera: 2023ko irailaren 4an).

[8]Horri buruzko intzidentzia bat ireki genuen 2020ko irailean Github-en (`https://github.com/tensorflow/tensorflow/issues/43497`); hala ere, oraindik ez dugu erantzun sendorik jaso (azken sarrera: 2023ko irailaren 4an).

[9]Ikus `https://www.tensorflow.org/guide/intro_to_graphs` informazio gehiagorako (azken sarrera: 2023ko irailaren 4an).

3.5.2. FEMeko datuen sorkuntza

SciPy liburutegia erabiliko dugu FEM ingurunean erabiltzen ditugun zabaltze-matrizeak, eta ekuazio linealen sistemaren matrizeak eta aurrebaldintzatzaileak eraikitzeko. Sakabanatuak diren sistema linealak ebazteko ere liburutegi bera erabiliko dugu. Geometria konplexuagoetarako, eraginkorragoak diren software-plataformak ere badaude, matrizeen, aurrebaldintzatzaileen eta zabaltze-operadoreen mihiztapena edo eraikuntza egiteko gai direnak (adibidez, FEniCS [8]). Horiek elementu finituen sistema konplexuagoetarako erabiltzea gomendatzen dugu (adibidez, 2D eta 3D problemetarako). Nolanahi ere, eragiketa horiek NNaren entrenamendua baino lehen egiten dira, eta, beraz, ez dute eraginik ereduaren optimizazio-denboran.

3.5.3. Aurrebaldintzatzaileen mihiztadura eta ekintzak

Idealki, aurrebaldintzatzaileak sistemako matrizeen blokekako LU deskonposaketa gisa kudeatu beharko genituzke, eta haien ekintzak galeren ebaluazio bakoitzean kalkulatu beharko genituzke, alderantzizko ordezkapen algoritmo bat erabiliz [5, 158]. Berriro, TF2-ak ez dauka ekintza horiek ebaluatzeko gai den tresna baliokiderik. Hori dela eta, mihiztatutako aurrebaldintzatzaileak erabiliko ditugu.

3.6. Zenbakizko esperimentuak

DeepFEMaren errendimendua aztertzeko, esperimentu batzuk gauzatuko ditugu. 3.6.1., 3.6.2. eta 3.6.3. ataletan esperimentu ez-parametrikoak aztertuko ditugu, eta 3.6.4. atalean, berriz, bi problema parametriko aztertuko ditugu. Esperimentu guztiak $(0, 1)$ espazioko eremuan egingo dira, Dirichlet eta Neumann muga-baldintzekin, 0n eta 1en, hurrenez hurren.

3.6.1. Adibide erraz bat

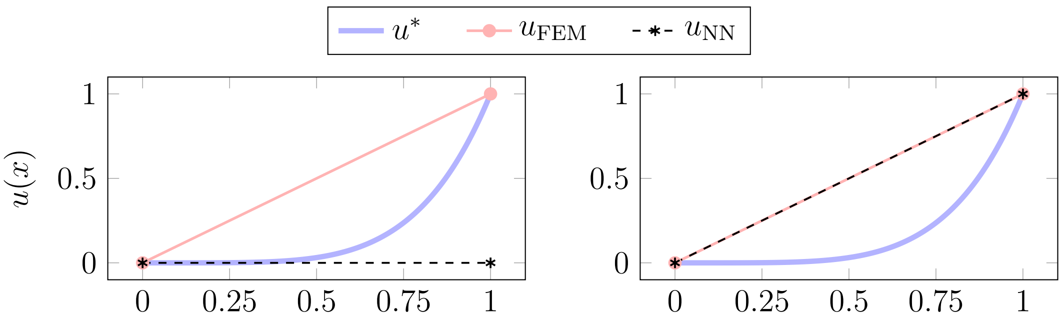
Izan bedi

$$\begin{cases} -u'' = -20x^3, \\ u(0) = 0, u'(1) = 5, \end{cases} \tag{3.25}$$

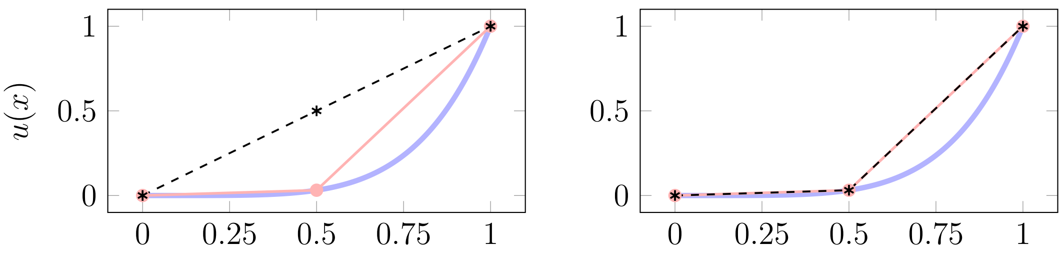
non bere soluzio analitikoa $u^*(x) = x^5$ baita. Hasieran, problema hau elementu bateko diskretizazio-sare bat erabiliz ebatziko dugu DeepFEMarekin. Gero, diskretizazio-sarea hamar aldiz finduko dugu (hamaika urrats, hau da, $1 \leq s \leq 11$), 1.024 elementudun diskretizazio-sare gainfindura iritsi arte. Lehen esperimentu honetan, proposatutako metodoa aztertuko dugu hainbat fintze eginez. Bloke entrenagarri bakoitzak neurona eta geruza bakarra ditu (gogoratu 3.2. atala). ReLU aktibazio funtzioak erabiliko ditugu bloke entrenagarri bakoitzean.

3.8. irudiak NNaren lehenengo lau urratsetako iragarpenak erakusten ditu. Urrats horietako bakoitzean, ereduak iragarpen zakarra diskretizazio-sare fineraino hedatzen du. Orduan, eredua entrenatzen da diskretizazio-sare finean hurbilketa zehatz bat lortzen den arte. Emaitzek zehaztasun handia erakusten dute lehen lau urratsetan.

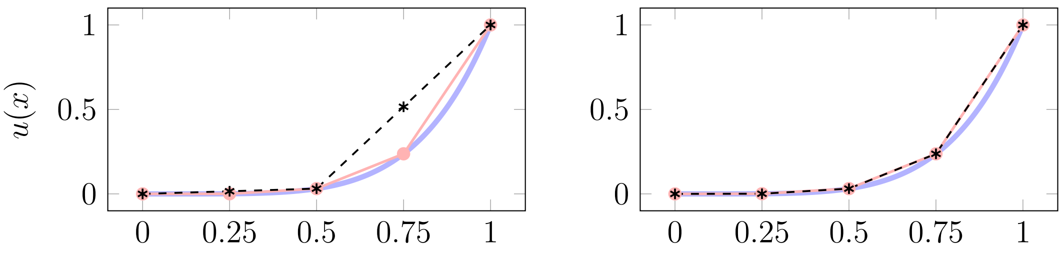
3.9a. irudiak galera-funtzioak lehenengo lau urratsetan duen eboluzioa erakusten du, galera-funtzio hori energia-normarekin bat datorrenean. 3.9b. irudiak galera-funtzioaren eboluzio osoa erakusten du (hamaika urratsetan zehar). Adibide horrek ikaskuntza sakonak baldintza optimaletan dituen konbergentzia-portaerak eta mugak agerian uzten ditu; adibidez, matrizearen alderantzizkoa galera-funtzioaren parte denean. Galera-funtzioan ikusten diren jauzi bertikalek iragarpenaren diskretizazio-sare zakarretik finerako hedapena adierazten dute (urrats batetik hurrengorakoa). Jauzi bakoitzaren ondoren, galera-funtzioaren konbergentzia zaratatsua ikusten da. Fase hau, Adam optimizatzailearen erabilerarekin lotuta dago. Hasieran, galera-funtzioa asko gutxitzea eragiten du; baina laster gelditzen da joera hori. Orduan, Adalr optimizatzailea erabiltzen dugu, galera-funtzioaren beherakada monotonoa erakutsiz. Fase bakoitzean, hasierako ikaskuntza-tasa da aurreko faseko galeraren lehenengo ebaluazioa bider 10^{-3} edo 10^{-2}, Adam edo Adalr erabiltzen ari garen arabera, hurrenez hurren. Gehie-



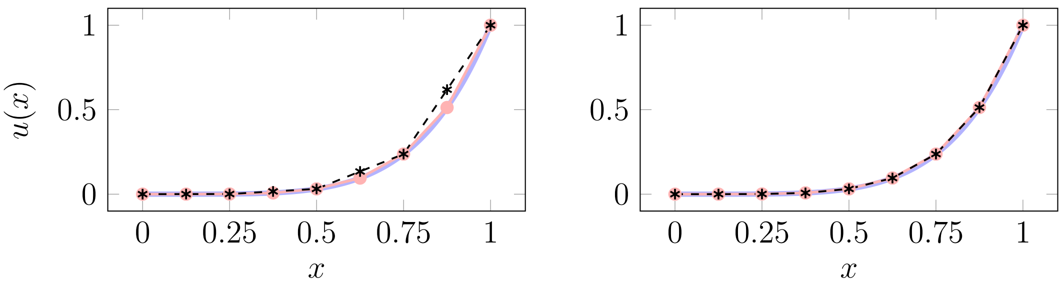
(a) $s = 1$ urratsa (entrenatu aurretik). **(b)** $s = 1$ urratsa (entrenatu ondoren).



(c) $s = 2$ urratsa (entrenatu aurretik). **(d)** $s = 2$ urratsa (entrenatu ondoren).



(e) $s = 3$ urratsa (entrenatu aurretik). **(f)** $s = 3$ urratsa (entrenatu ondoren).



(g) $s = 4$ urratsa (entrenatu aurretik). **(h)** $s = 4$ urratsa (entrenatu ondoren).

3.8. irudia: (3.25). problemaren lehenengo lau urratsetako DeepFEMeko iragarpenak. u^* soluzio zehatza da, u_{FEM} elementu finituetako soluzioa, eta u_{NN} DeepFEMaren iragarpena.

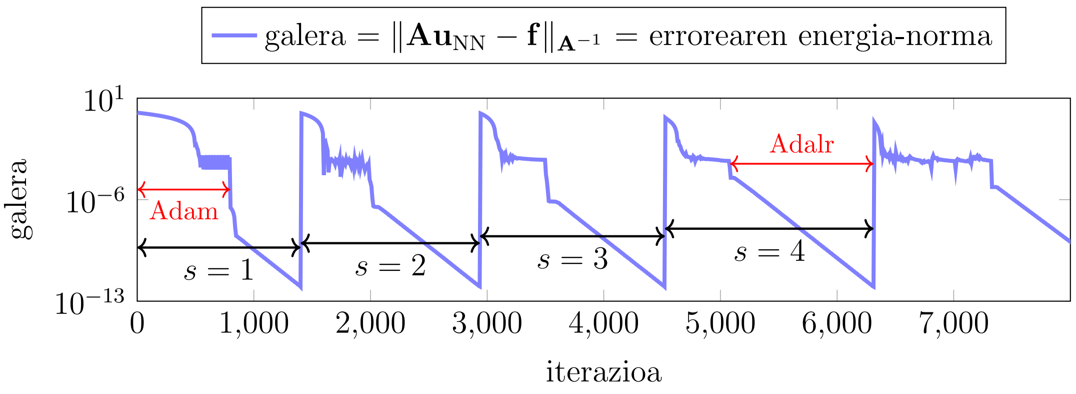
nez, 2.000 eta 4.000 iterazio ezarri ditugu Adam-entzako eta Adalr-entzako fase bakoitzeko exekuzioetan, hurrenez hurren. Hasieratik amaierarainoko entrenamenduak egiten ditugu NNan. Konbergentzia-desbideraketa nabaritzen dugu urrats-kopurua (eta diskretizazio-sarearen tamaina) handitzean. Nolanahi ere, azken errorea 10^{-8} baino txikiagoa da urrats guztietan zehar (ikus 3.9c. irudia).

3.6.1.1. Hasieratik amaierarainoko entrenamendua vs. geruzaz geruzakoa

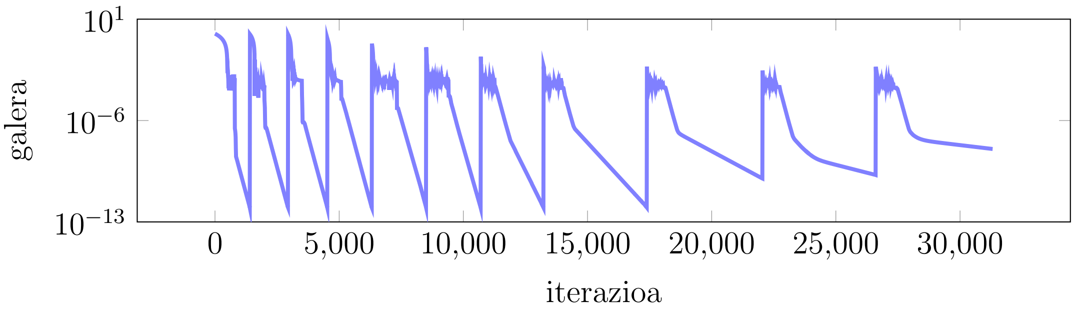
Eraikuntzarengatik, DeepFEMaren geruza bakoitzak diskretizazio-sarearen elementu bakoitzarekin elkartutako oinarriko funtzioekin lotutako koefizienteak iragartzen ditu. Diskretizazio-sareak zenbat eta finagoak izan, orduan eta lokalagoak dira oinarriko funtzioei lotutako euskarriak. Alde batetik, hasieratik amaierarainoko entrenamenduak koefizienteen hierarkia osoaren entrenagarritasuna ahalbidetzen du. Beste alde batetik, geruzaz geruzako entrenamenduak, bakarrik azken diskretizazio-sarearekin (finena dena) lotuta dauden koefizienteen entrenagarritasuna hornitzen du. 3.10a. irudiak galera-funtzioaren konbergentzia erakusten du, goiko baldintza berberetan, baina geruzaz geruzako entrenamendua egiten denean. Ikusten den moduan, konbergentzia desorekatu egiten da iterazio kopurua handitzen den heinean, hasieratik amaierarainoko entrenamenduarekin konparatuta (gogoratu 3.9b. irudia). Hori gertatzen da galera-funtzioak errorearen gradientea dakarrelako energia-norma erabiltzen denean. 3.10b. irudiak errore-funtzioa irudikatzen du, ia konstantea dena. Horrela, errorearen deribatua ia zero da. Orduan, erronka izaten da errorearen energia-norma minimizatzea, lokalak diren euskarri funtzioei lotutako koefizienteak doituz. Horrek, askotan, errorearen deribatua handitzea dakar. Konbergentzia azkartu egiten da euskarri globaleko eta lokaleko oinarriko funtzioak erabiltzen direnean, multidiskretizazio-sareetan oinarritutako metodoekin bezala (ingelesetik, *multigrid methods*) [37].

Gradienteei jaramonik egiten ez dien norma bat aukeratzen badugu, adibidez, L^2-norma, euskarri lokaleko oinarriko funtzioei lotutako entrenamenduak emaitza bikainak ematen ditu (geruzaz geruzako entrenamendua egitean), 3.11a. irudian ikusten den moduan. Hala ere, ekuazio diferentzialak ebazten direnean ez da gomendagarria L^2-norma erabilita optimizatzea.

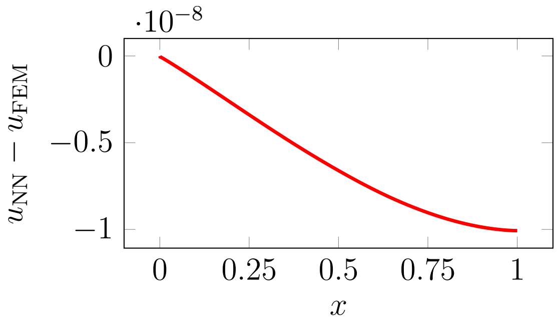
Nahiz eta hasieratik amaierarainoko entrenamendua aukerarik onena izan hondarren/erroreen gradientea darabilten galera-funtzioetan, galera-funtzioaren desbideraketa ikusten dugu iterazioek aurrera egin ahala (gogoratu 3.9b. irudia). Pentsatzen dugu hori normaren aukeraketarekiko independentea dela, baina aldagai entrenagarri askoren koexistentzia kontrajarriaren ondorioa dela. Gertaera hori argitzeko, L^2-normaren goiko kasua aztertuko dugu, non geruzaz geruzako entrenamendua nahikoa baitzen konbergentzia bikaina lortzeko, eta hasieratik amaieraraionoko entrenamendua egingo dugu. 3.11b. irudiak galera-funtzioaren konbergentzia desbideraketa erakusten du.



(a) Galera-funtzioaren eboluzioa lehenengo lau urratsetan.

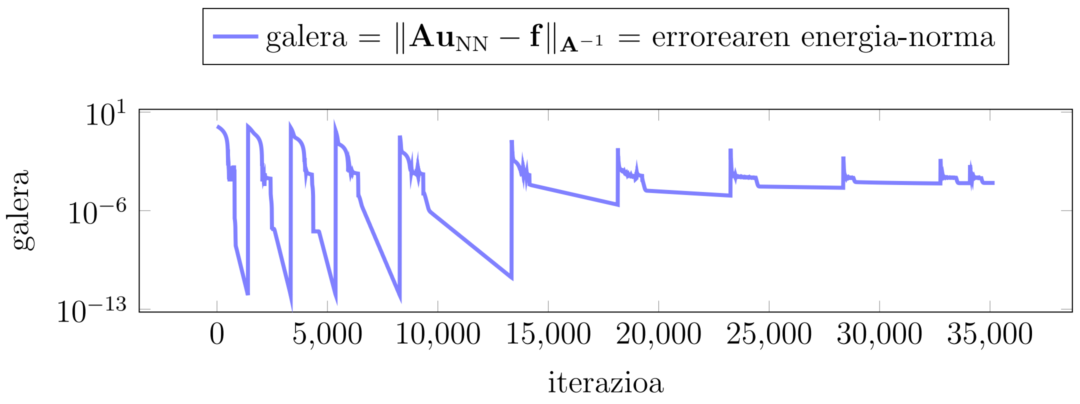


(b) Galera-funtzioaren eboluzioa hamaika urratsetan.

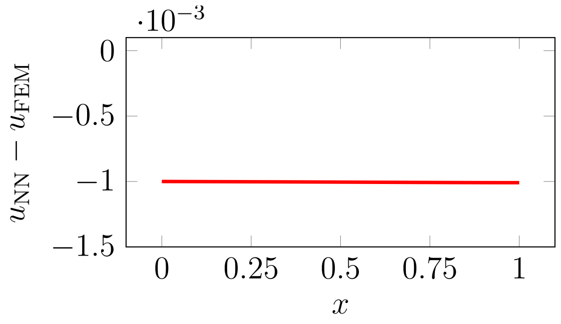


(c) Errore-funtzioa entrenamenduaren amaieran.

3.9. irudia: Hasieratik amaierarainoko entrenamendua DeepFEMarentzako (3.25). probleman. Galera-funtzioa errorearen energia-norma da.



(a) Galera-funtzioaren eboluzioa hamaika urratsetan.



(b) Errore-funtzioa entrenamenduaren amaieran.

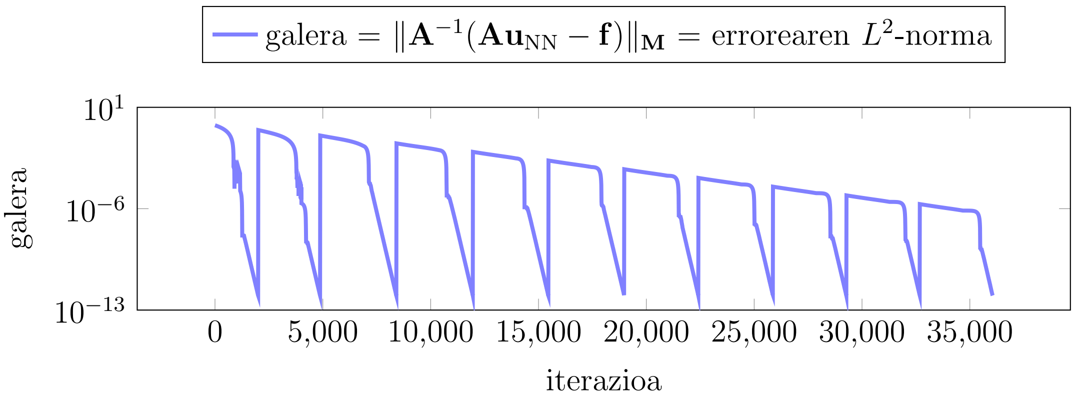
3.10. irudia: Geruzaz geruzako entrenamendua DeepFEMean galera-funtzioan energia-norma erabiltzen denean (3.25). probleman.

Jarraian, bakarrik hasieratik amaierarainoko entrenamenduak egingo ditugu.

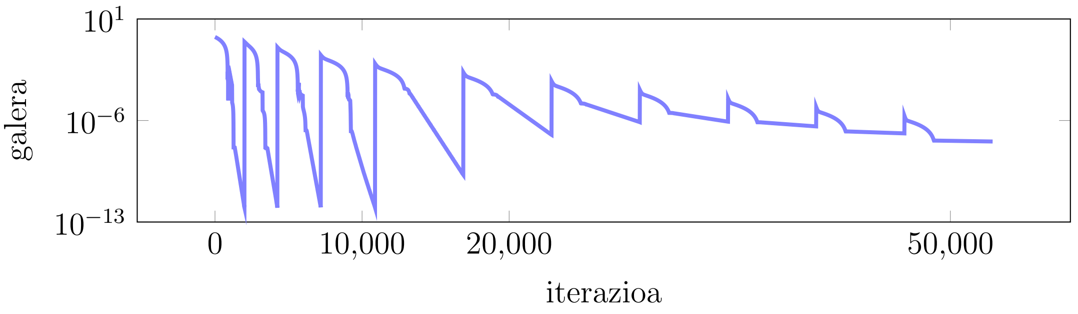
3.6.1.2. Aurrebaldintzatzaileen ekintzak

Orain, (3.19). ekuazioak adierazten duen galera-funtzioa aztertuko dugu hiru aurrebaldintzatzailerekin: (a) **P** identitate-matrizea izanik (3.12a. irudia); (b) **P** blokekako Jacobi aurrebaldintzatzailea, bi tamainako blokeekin (3.12b. irudia); eta (c) **P** blokekako Jacobi aurrebaldintzatzailea egokitutako tamainako blokeekin, haien tamaina diskretizazio-sarearen elementu kopuruaren erdia izanik (3.12c. irudia). Kasu guztietan, galera-funtzioaren eboluzioa eta errorearen energia-normarena erakusten ditugu.

Zenbait ezberdintasun ikusten ditugu errorearen energia-normaren eta galera-funtzioaren artean; espero bezala, galera-funtzioa handitu egiten da diskretizazio-sarea handitzen denean. Jacobiren blokea zenbat eta handiagoa izan, galera-funtzioaren eta errorearen normaren arteko ezberdintasuna txikiagoa da. Gaine-



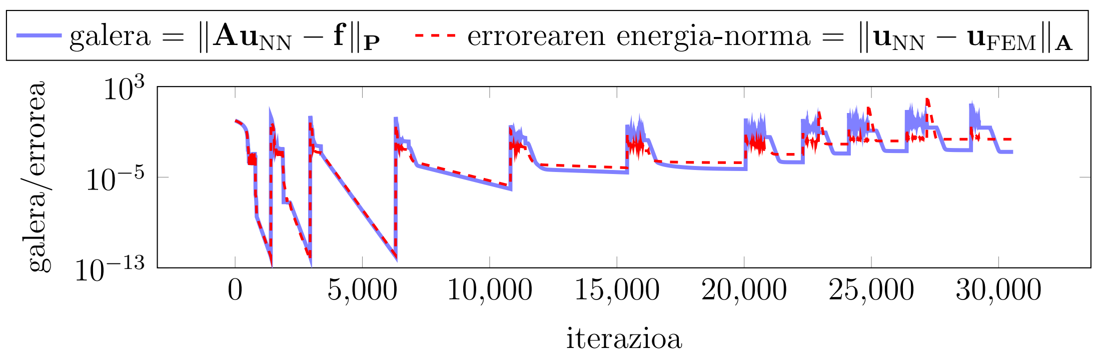
(a) Galera-funtzioaren eboluzioa geruzaz geruzako entrenamenduak egiten direnean.



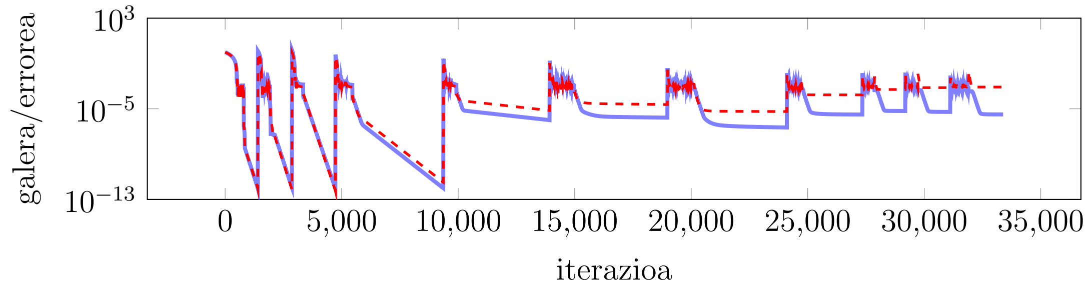
(b) Galera-funtzioaren eboluzioa hasieratik amaierarainoko entrenamenduak egiten direnean.

3.11. irudia: DeepFEMaren entrenamenduak (3.25). problemarentzat galera-funtzioan L^2-norma erabiltzean.

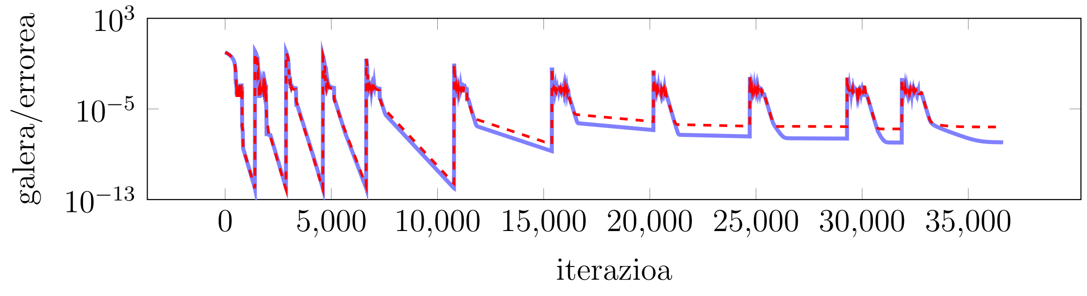
ra, galera-funtzioa errorearen energia-normatik zenbat eta urrunago egon, galera-funtzioak arinago lortzen du konbergentzia (konparatu 3.12d eta 3.12e. irudietako erroreak). Horrek pentsarazten digu energia-normako erroreak induzitutako galera-funtzioa sinplifikatutako beste aldaera batzuk baino ganbilagoa dela aldagaiekiko, besteak beste hondar bektorearen 2-normak induzitutako galera-funtzioa baino ganbilagoa.



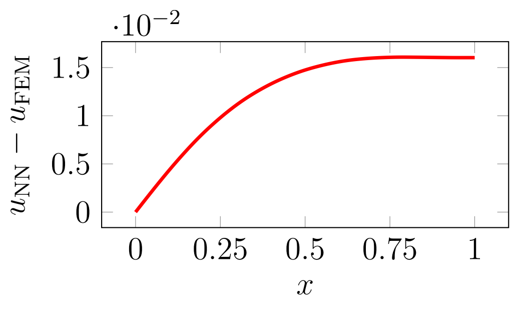
(a) Aurrebaldintzatzaile barik, hau da, $\mathbf{P} = \mathbf{I}$.



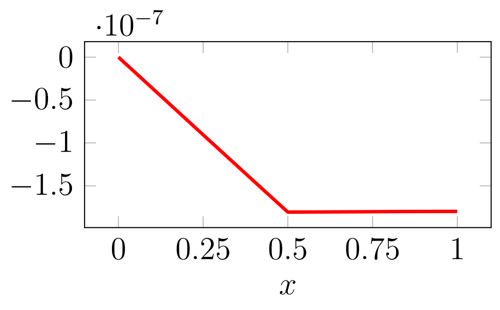
(b) Bi tamainako blokeak erabilita.



(c) Diskretizazio-sarearen elementu kopuruaren erdiko tamainako blokeak erabilita.



(d) (a). kasuaren errore-funtzioa.



(e) (c). kasuaren errore-funtzioa.

3.12. irudia: Hasieratik amaierarainoko DeepFEMaren entrenamenduak (3.25). problemarentzat tamaina ezberdineko blokedun Jacobi aurrebaldintzatzaileak erabilita.

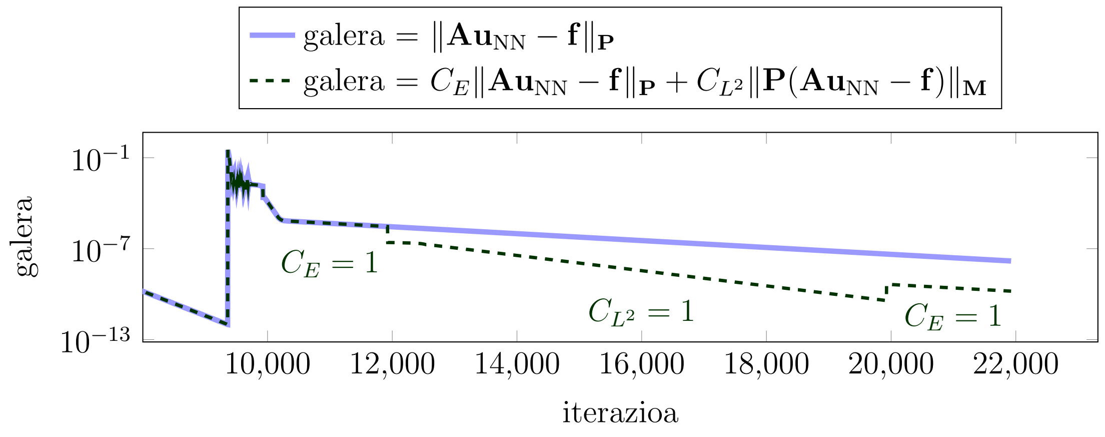
3.6.1.3. Normaren trukaketa entrenamenduan zehar

Aukeratzen dugun normaren arabera, galera-funtzioaren ganbiltasun mota ezberdinak agertuko dira DeepFEMaren aldagaiekiko. Sarri gertatzen da aldagaien domeinuaren zenbait zatitan galera-funtzio bat beste batzuk baino ganbilagoa izatea, eta horrek zuzeneko eragina du konbergentziaren optimizatzailean. Gelditzea saihesteko, eta ganbiltasuna hobetzeko, optimizazioan zehar norma aldatzea proposatzen dugu. Horretarako, esaterako, honako era honetan definituko dugu norma:

$$\mathcal{L}(\sigma, \alpha; \theta) = C_E \|\mathbf{A}\mathbf{u}_{\mathrm{NN}} - \mathbf{f}\|_{\mathbf{P}} + \ C_{L^2} \|\mathbf{P}(\mathbf{A}\mathbf{u}_{\mathrm{NN}} - \mathbf{f})\|_{\mathbf{M}}, \tag{3.26}$$

non $C_E, C_{L^2} \in \{0, 1\}$ balio ezberdinak baitira, eta elkarrekin trukatuko dira konbergentzia gelditzea gertatzen baldin bada.

Goian aipaturiko ideia argitzeko, 11b. irudiaren kasua aztertuko dugu. Galera-funtzio berbera mantenduko dugu lehenengo lau urratsetan. Gero, bosgarren urratsaren galera-funtzioa ezartzeko bi aldaera hartu ditugu kontuan: (a) aurreko urratseko galera-funtzio berbera ($C_E = 1$ ezarriz (3.26). ekuazioan) baina 12.000 iteraziorekin; eta (b) (3.26). ekuazioan $C_E = 1$ ezarriz 2.000 iteraziotan, gero $C_{L^2} = 1$ ezarri dugu beste 8.000 iteraziotan, eta azkenean $C_E = 1$ ezarpenera bueltatu gara azkenengo 2.000 iterazioetan. Nahiz eta aldaera bietako iterazio kopuru osoa berdina izan, galera-funtzioaren balore txikiagoa lortzen dugu bosgarren urratsa amaitu ondoren norma aldatzen dugunean (10^{-10} inguru) aldaketarik egiten ez denean baino (10^{-8} inguru) —ikus 3.13. irudian—. Galera-funtzioaren eboluzioaren malda altuagoa da L^2-norma darabilgunean, eta horrek galera baxuagoko balio batetik abiatzea ahalbidetzen du energia-normara itzultzean, entrenamendu osoan zehar berbera mantentzen denean baino.



3.13. irudia: Energia-normaren entrenamendu luzea vs. energia- eta L^2-normaren arteko trukea 3.12b. irudiaren bosgarren ($s = 5$) urratsaren erdian.

3.6.2. Soluzio sinusoidalak Poisson and Helmholtz ekuazioetan

Izan bitez

$$\begin{cases} -u'' = 100\pi^2 \sin(10\pi x), \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \qquad \begin{cases} -u'' - 100\pi^2 u = 0, \\ u(0) = 0, u'(1) = 10\pi. \end{cases} \tag{3.27}$$

Hemen, soluzio zehatz berdina duten bi problema ezberdin ditugu (lehenengoa simetrikoa eta positibo-definitua da, eta bigarrena, berriz, indefinitua):

$$u^*(x) = \sin(10\pi x). \tag{3.28}$$

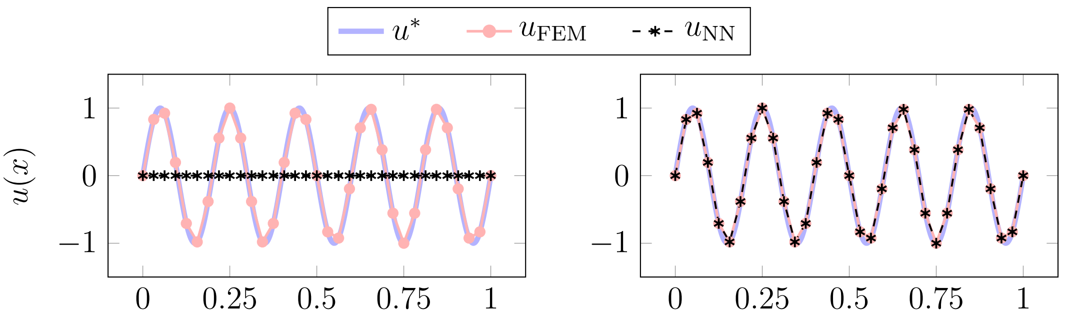
Biak ebatziko ditugu DeepFEM erabilita, eta galera-funtzioa aurrebaldintzatutako hondarraren H^1-norma bezala hautatuko dugu, hau da,

$$\mathcal{L}(\theta; \sigma, \alpha) = \|\mathbf{P}(\mathbf{A}\mathbf{u}_{\text{NN}} - \mathbf{f})\|_{\mathbf{K}+\mathbf{M}}. \tag{3.29}$$

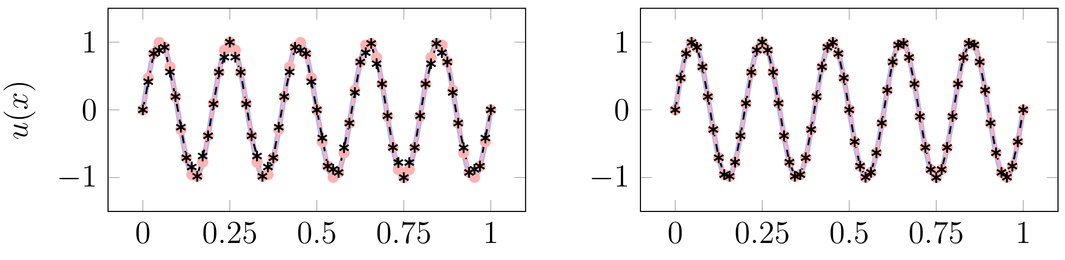
32 elementuko diskretizazio-sare batekin hasiko gara eta hiru fintze egingo ditugu. $\mathbf{P}$ aurrebaldintzatzailea hautatuko dugu 32 elementurekin eta $1 \leq s \leq 4$ urratsetan. Optimizatzaileak eta NNaren arkitekturak 3.6.1. atalekoak dira.

3.14. eta 3.15. irudiek Poisson eta Helmholtz problemekin lotutako DeepFEM iragarpenak erakusten dituzte, hurrenez hurren. Poissonen, FEMaren soluzioek u^*-rekin bat egiten dute nodoetan, baina hori ez da gertatzen Helmholtz ekuazioaren kasuan.

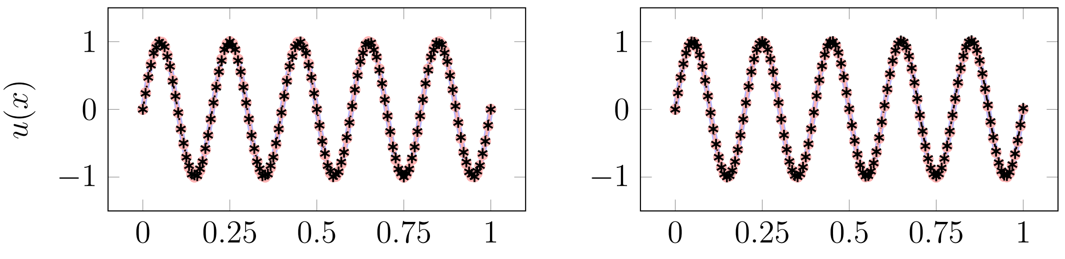
Aurrebaldintzatzaile gisa alderantzizko matrizea hartuz entrenatzen denean, DeepFEMak beherapen monotonikoa erakutsiz konbergitzen du. Hala ere, alderantzizkoa ez den aurrebanditzatzaile bat hartzen denean, konbergentzia gelditu egiten da pare bat magnitude-orden murriztu ondoren (ikus 3.16a. eta 3.16b. irudiak). Adalr optimizatzailearen entrenamendu-faseetan, galera-funtzioa batbatean murrizten da lehen iterazioetan, baina gero lautu egiten da, H^1-normari dagokion errore murrizketa hutsala eginez. Bi esperimentuetan, 10^{-3} inguruko erroreak lortzen ditugu (ikus 3.16c. eta 3.16d. irudiak). Emaitza horiek erakusten dute nolabaiteko zehaztasun-maila (baina ez handia) duten soluzio hurbilduak lor daitezkeela, ziur asko galerak dituen ganbiltasun ezagatik (gogoratu 2.3. atala).



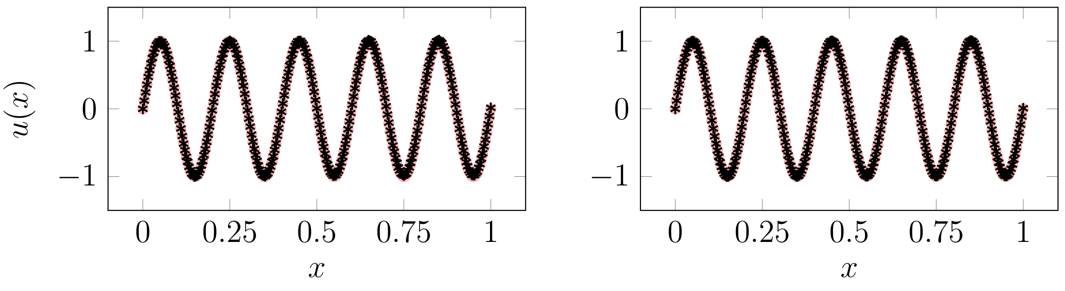
(a) $s = 1$ urratsa (entrenatu aurretik). **(b)** $s = 1$ urratsa (entrenatu ondoren).



(c) $s = 2$ urratsa (entrenatu aurretik). **(d)** $s = 2$ urratsa (entrenatu ondoren).

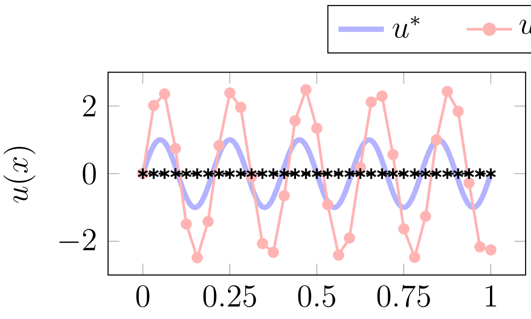


(e) $s = 3$ urratsa (entrenatu aurretik). **(f)** $s = 3$ urratsa (entrenatu ondoren).

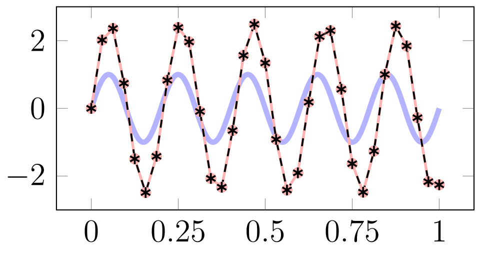


(g) $s = 4$ urratsa (entrenatu aurretik). **(h)** $s = 4$ urratsa (entrenatu ondoren).

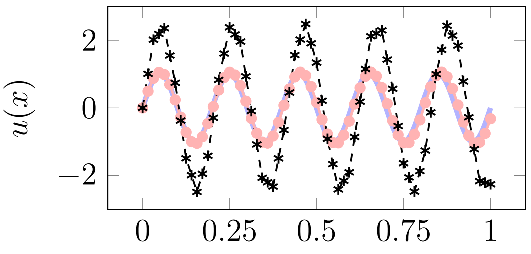
3.14. irudia: Poisson (3.27). probleman lortutako iragarpenak lau urratsetan zehar. u^* soluzio zehatza da, u_{FEM} elementu finituetako soluzioa, eta u_{NN} DeepFEMaren iragarpena.



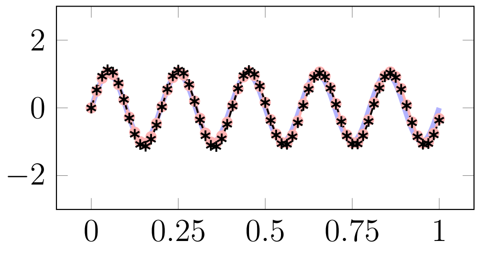
(a) $s = 1$ urratsa (entrenatu aurretik).



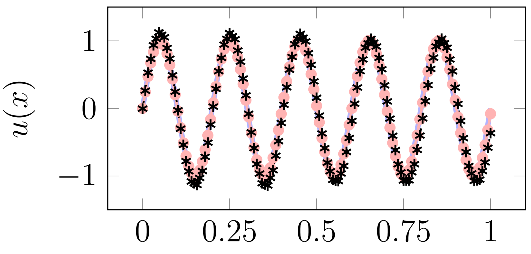
(b) $s = 1$ urratsa (entrenatu ondoren).



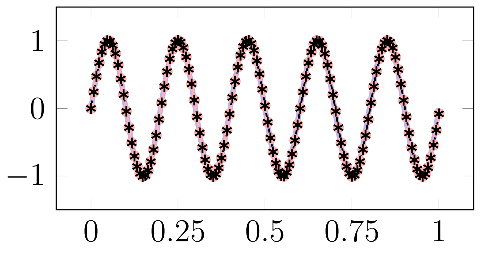
(c) $s = 2$ urratsa (entrenatu aurretik).



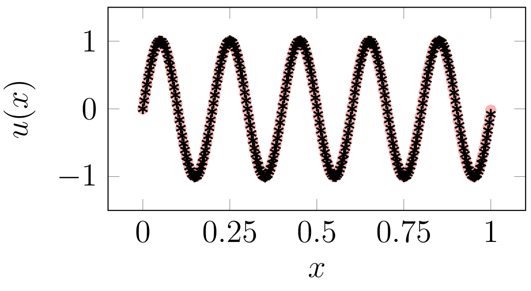
(d) $s = 2$ urratsa (entrenatu ondoren).



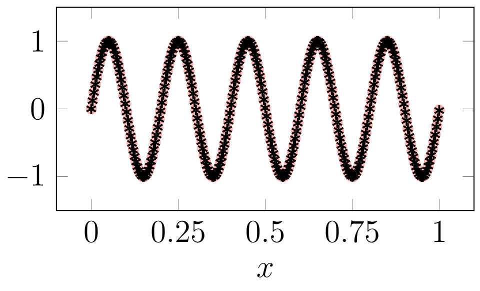
(e) $s = 3$ urratsa (entrenatu aurretik).



(f) $s = 3$ urratsa (entrenatu ondoren).

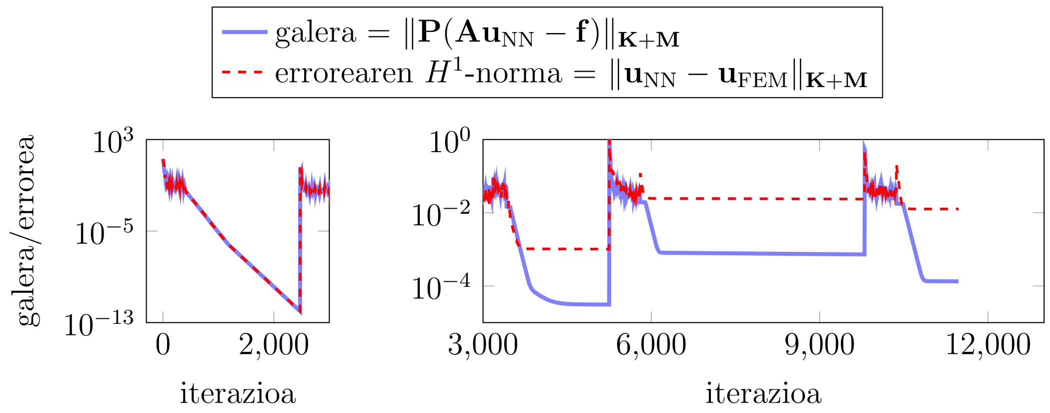


(g) $s = 4$ urratsa (entrenatu aurretik).

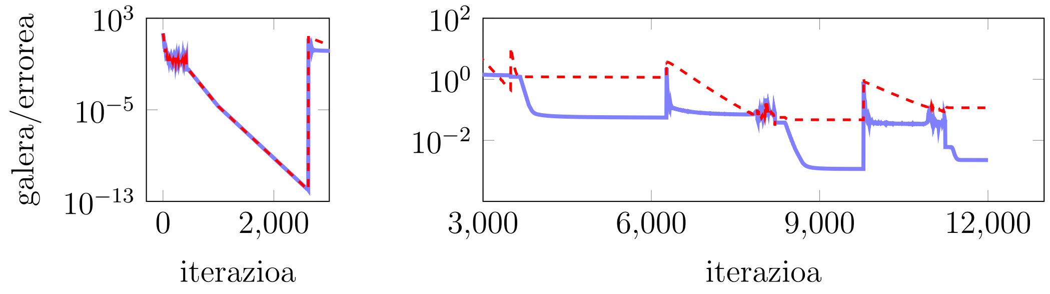


(h) $s = 4$ urratsa (entrenatu ondoren).

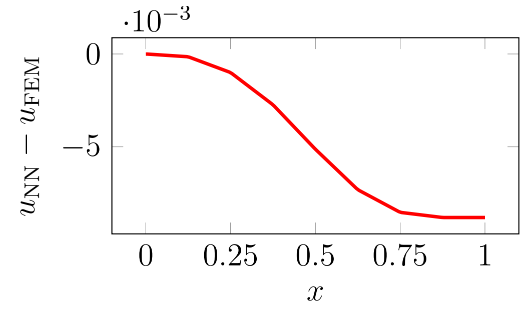
3.15. irudia: Helmholtz (3.27). probleman lortutako iragarpenak lau urratsetan zehar. u^* soluzio zehatza da, u_{FEM} elementu finituetako soluzioa, eta u_{NN} DeepFEMaren iragarpena.



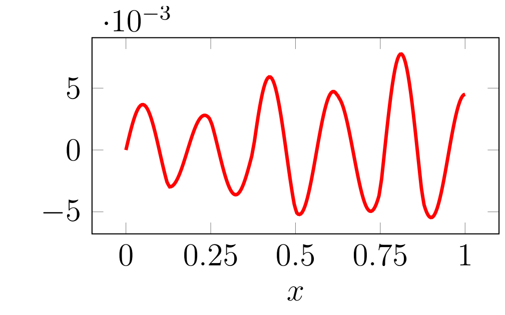
(a) Poisson (3.27). probleman galeraren eta errorearen H^1-normaren eboluzioak.



(b) Helmholtz (3.27). probleman galeraren eta errorearen H^1-normaren eboluzioak.



(c) Errore-funtzioa entrenamenduaren amaieran (Poisson).



(d) Errore-funtzioa entrenamenduaren amaieran (Helmholtz).

3.16. irudia: Hasieratik amaierarainoko entrenamenduak DeepFEMean hiru urratsetan zehar (3.27). probleman, 32 tamainako Jacobi blokedun aurrebaldintzatzaileak erabilita.

3.6.3. Soluzio sinusoidala zatika konstanteak diren koefizienteak erabiliz

Helmholtzen ekuazioari jarraiki, maiztasun aldakorrak gehitzen ditugu hedapen-domeinuan, zatika konstanteak diren koefizienteak kontuan hartuta, honela:

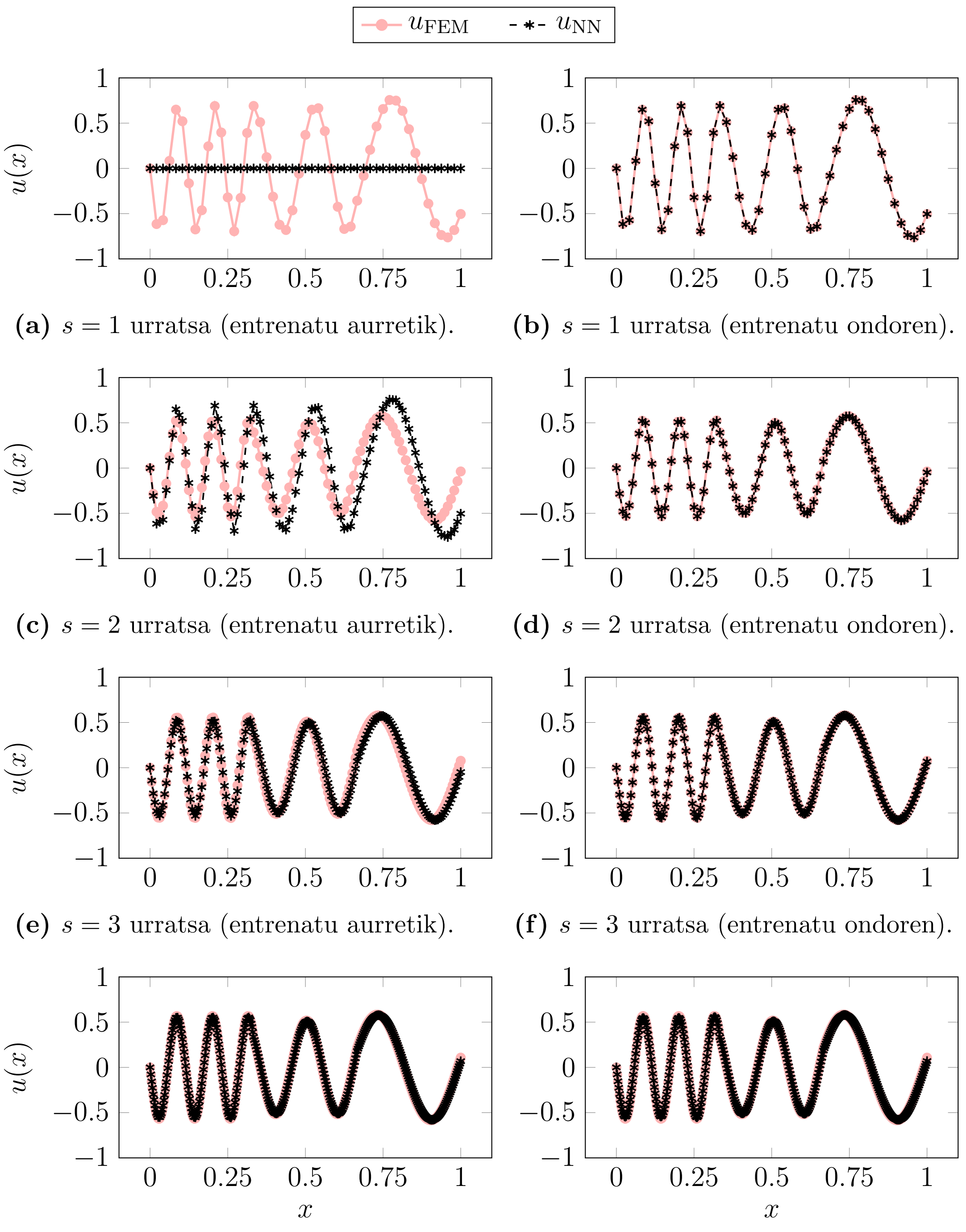
$$\begin{cases} -(\sigma u')' + \alpha u = 0, \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \tag{3.30}$$

non

$$\sigma = \begin{cases} 1, & 0 < x < 1/3 \text{ bada}, \\ 2, & 1/3 < x < 2/3 \text{ bada}, \\ 3, & 2/3 < x < 1 \text{ bada}, \end{cases} \quad \alpha = \begin{cases} -3000, & 0 < x < 1/3 \text{ bada}, \\ -2000, & 1/3 < x < 2/3 \text{ bada}, \\ -1000, & 2/3 < x < 1 \text{ bada}. \end{cases} \tag{3.31}$$

Galera-funtziorako H^1-norma erabiltzen dugu. 48 elementu dituen diskretizazio-sarearekin hasiko gara, eta hiru fintze uniforme egingo ditugu. 48 tamainako blokedun **P** aurrebaldintzatzailea hautatuko dugu diskretizazio-sarearen lehen bi fintzeetan, eta 96 tamainako blokeak dituena azken bietan. Optimizatzaileak eta NNaren arkitekturak 3.6.1. atalekoak dira berriro.

3.17. irudiak DeepFEMaren iragarpenak erakusten ditu eta 3.18a. irudiak galera-funtzioaren eboluzioa erakusten du entrenamenduan zehar. Alderantzizkoa erabiltzen denean aurrebaldintzatzaile gisa, entrenamendua ondo doa, baina hurrengo urratsetan galeraren geldialdia ikusten da, lehen gertatzen zen bezala. Gainera, galeraren eragin beherakorra dago errorearen H^1-norman, azken entrenamendu-urratsean konstante mantentzen dena galera bi magnitude-ordenatan jaisten den bitartean. 3.18b. eta 3.18c. irudiek hirugarren eta laugarren urratsen amaierako errore-funtzioak erakusten dituzte, hurrenez hurren.



(a) $s = 1$ urratsa (entrenatu aurretik).

(b) $s = 1$ urratsa (entrenatu ondoren).

(c) $s = 2$ urratsa (entrenatu aurretik).

(d) $s = 2$ urratsa (entrenatu ondoren).

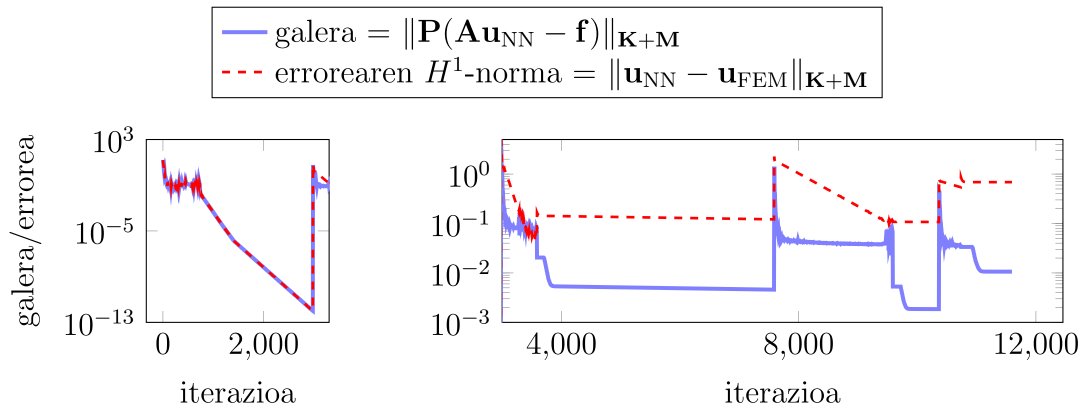
(e) $s = 3$ urratsa (entrenatu aurretik).

(f) $s = 3$ urratsa (entrenatu ondoren).

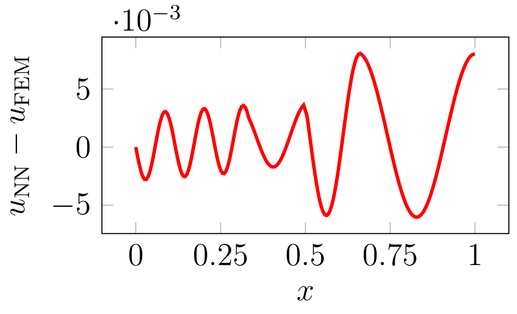
(g) $s = 4$ urratsa (entrenatu aurretik).

(h) $s = 1$ urratsa (entrenatu ondoren).

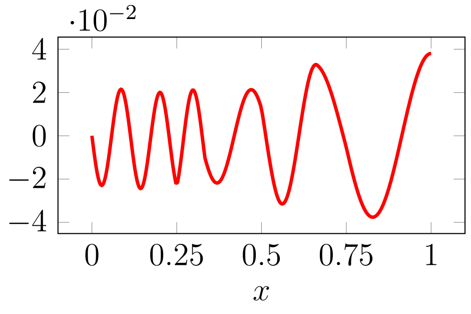
3.17. irudia: (3.30). probleman lortutako iragarpenak lau urratsetan zehar. u^* soluzio zehatza da, u_{FEM} elementu finituetako soluzioa, eta u_{NN} DeepFEMaren iragarpena.



(a) Galeraren eta errorearen H^1-normaren eboluzioak Helmholtz (3.30). probleman.



(b) Errore-funtzioa $s = 3$ urratsaren amaieran.



(c) Errore-funtzioa $s = 4$ urratsaren amaieran.

3.18. irudia: Hasieratik amaierarainoko entrenamenduak DeepFEMean lau urratsetan zehar (3.30). probleman. 48, 48, 96 eta 96 tamainako Jacobi blokedun aurrebaldintzatzaileak erabili dira 1., 2., 3. eta 4. urratsetan, hurrenez hurren.

3.6.4. Mutur-balioen problema parametrikoak

Atal honetan, DeepFEMa bere aldaera parametrikora hedatuko dugu: NNa entrenatu dugu FEMaren soluzioak diskretizazio-sare batetik bestera ikas ditzan, muga-baldintza finkoak dituen PDE koefiziente familia baterako. Esperimentutzat hartuko ditugu α koefizienteak bakarrik portaera parametriko konstanteak dituen kasuak.

Izan bedi

$$\begin{cases} -u'' + \alpha u = 0, \\ u(0) = 0, u'(1) = 2\pi. \end{cases} \tag{3.32}$$

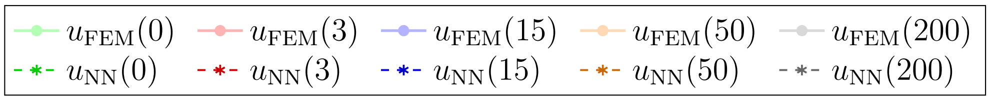
Problema hau ebazteko, diskretizazio-sarea hiru aldiz finduko dugu. Ausazko datu-baseak aukeratuko ditugu α koefizientearentzat. Entrenamendua datu-base osoan egingo da NNan, multzokaketako zatiketa egin barik. Aukeratutako galera-funtzioak norma aplikatzen dio aurrebaldintzatzailedun hondar bakoitzari, zeina datu lagin bakoitzaren mendekoa baita. Zehazki, lagineko galera-funtzioen batezbesteko batuketa batekiko optimizatuko dugu —gogoratu (3.15). ekuazioa—.

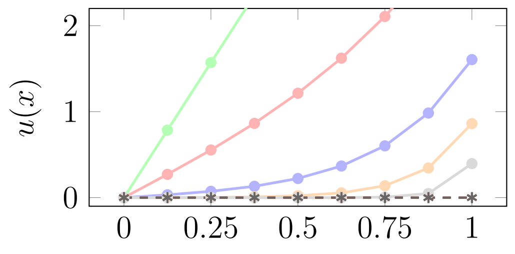
3.6.4.1. Erreakzio-difusio ekuazio parametrikoa: $0 < \alpha < 200$

Kasu honetako soluzio analitikoa honako hau da: $u^*(x) = C(e^{\sqrt{\alpha}x} - e^{-\sqrt{\alpha}x})$, $C = \frac{2\pi e^{\sqrt{\alpha}}}{\sqrt{\alpha}(e^{2\sqrt{\alpha}}+1)}$ izanik, eta $u^*(x) \to 2\pi x$ betetzen da $\alpha \to 0$ denean. Geruza bakarreko sakontasuna eta 20 neuronen zabalera ezarriko dugu NNaren bloke entrenagarri bakoitzean. Zortzi elementuko diskretizazio-sare batekin hasiko gara DeepFEMarekin, eta 64 elementuko diskretizazio-sare batekin amaituko dugu. Urrats bakoitzean, aurrebaldintzatzaileak erabiliko ditugu zortzi tamainako blokeekin. Problema parametrikoa simetriko eta positibo-definitua denez, aurrebaldintzatzaileak sortzen duen norma erabiliko dugu optimizazio prozesuan.

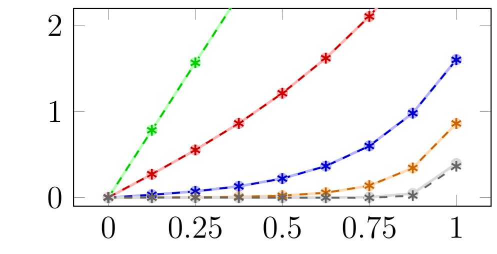
Ehun lagineko datu-base bat aukeratuko dugu NNa entrenatzeko. Entrenamenduaren ondoren, NNaren errendimendua aztertzeko, honako proba datuak (ingelesetik, *test data*) hautatuko ditugu: $\{0, 3, 15, 50, 200\}$. Probarako datuek ez dute entrenamenduan parterik hartuko. 3.19. irudiak NNaren iragarpen parametrikoak irudikatzen ditu proba datuetan ebaluatuta, eta diskretizazio-sare bakoitzetik besterako adaptatibitatea erakusten dute. α-ren balio guztietarako NNaren portaera ona ikusten dugu. 3.1. taulak galera-funtzioaren eta errorearen energia-normak erakusten ditu proba datuetan ebaluatuta, aurrebaldintzatutako entrenamendu honen amaieran.

3.20a. irudiak galera-funtzioaren eta errorearen energia-normaren eboluzioak erakusten ditu. Ohartzen gara galera-funtzioaren beherakada errorearen energia-norman baino handiagoa dela, espero zen bezala. Aurrebaldintzatzaileen blokeen tamaina urratsez urrats handitzen badugu (adibidez, 8, 8, 16 eta 32 tamainekin

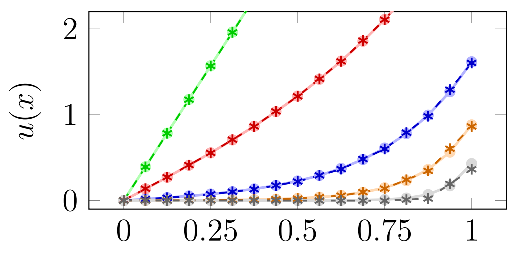




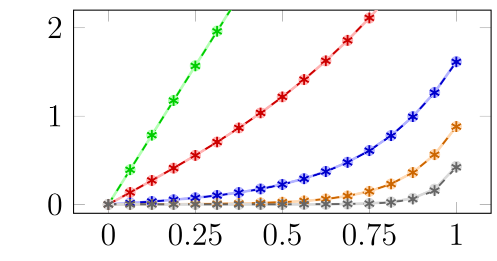
(a) $s = 1$ urratsa (entrenatu aurretik).



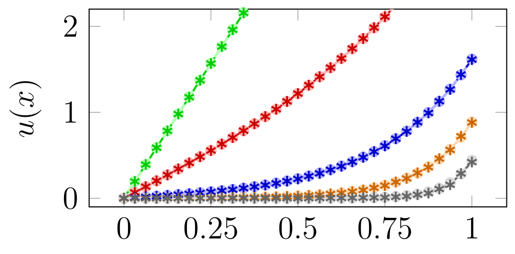
(b) $s = 1$ urratsa (entrenatu ondoren).



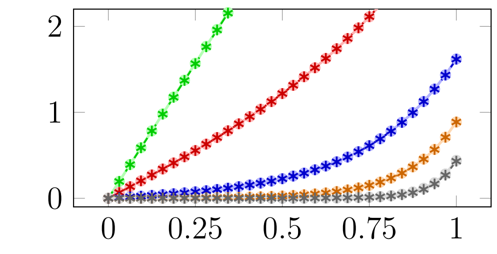
(c) $s = 2$ urratsa (entrenatu aurretik).



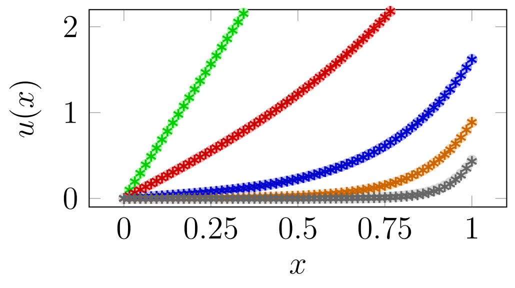
(d) $s = 2$ urratsa (entrenatu ondoren).



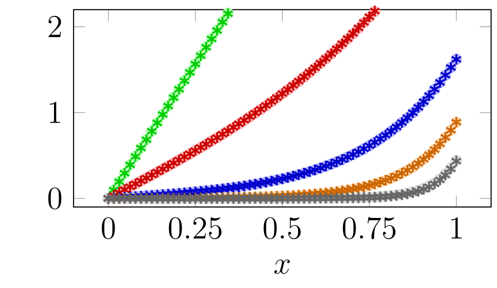
(e) $s = 3$ urratsa (entrenatu aurretik).



(f) $s = 3$ urratsa (entrenatu ondoren).



(g) $s = 4$ urratsa (entrenatu aurretik).



(h) $s = 4$ urratsa (entrenatu ondoren).

3.19. irudia: Lau urratsetan lortutako iragarpenak proba datu-basean (3.32). probleman, $0 < \alpha < 200$ denean. Galera-funtzioan energia-norma erabili dugu. $u_{\mathrm{FEM}}(\alpha)$ elementu finituetako soluzioa da eta $u_{\mathrm{NN}}(\alpha)$ DeepFEMaren iragarpena α parametroarentzako.

α	0	3	15	50	200
$\Vert\mathbf{A}(\alpha)\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{f}\Vert_{\mathbf{P}(\alpha)}$	0.0015	0.013	0.0046	0.012	0.082
$\Vert\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{u}_{\mathrm{FEM}}(\alpha)\Vert_{\mathbf{A}(\alpha)}$	0.0160	0.014	0.0061	0.014	0.095

3.1. taula: Hondarraren eta errorearen energia-normaren balioak proba datu-basearen lagin bakoitzerako entrenamenduaren amaieran, (3.32). probleman eta $0 < \alpha < 200$ denean. Zortzi tamainako blokeak erabili ditugu aurrebaldintzatzaileetan.

1., 2., 3. eta 4. urratsetan, hurrenez hurren), errorearen energia-norma eta galera-funtzioa antzera txikitzen dira (ikus 3.20b. irudia). 3.2. taulak galera-funtzioaren eta errorearen energia-normak erakusten ditu proba datuetan ebaluatuta, aurrebaldintzatutako entrenamenduaren amaieran.

α	0	3	15	50	200
$\Vert\mathbf{A}(\alpha)\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{f}\Vert_{\mathbf{P}(\alpha)}$	0.0022	0.0059	0.0043	0.0086	0.011
$\Vert\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{u}_{\mathrm{FEM}}(\alpha)\Vert_{\mathbf{A}(\alpha)}$	0.0053	0.0075	0.0058	0.0101	0.013

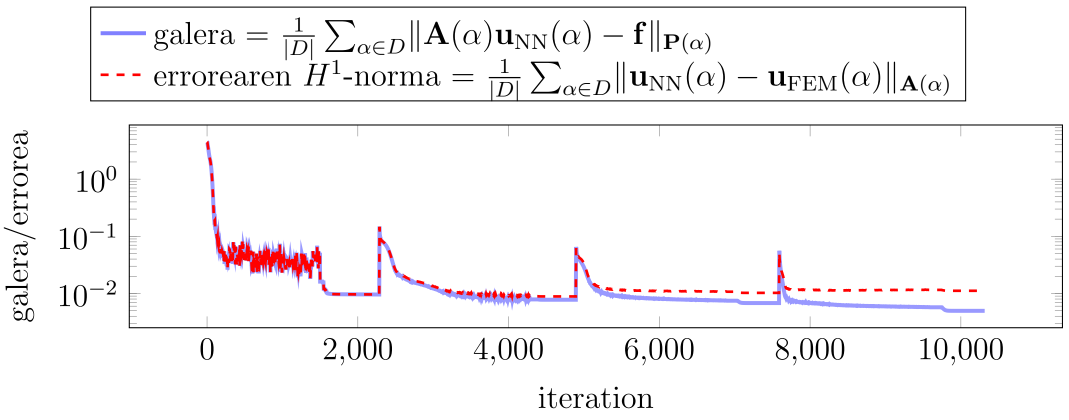
3.2. taula: Hondarraren eta errorearen energia-normaren balioak proba datu-basearen lagin bakoitzerako entrenamenduaren amaieran, (3.32). probleman eta $0 < \alpha < 200$ denean. Tamaina gorakorreko blokeak erabili ditugu aurrebaldintzatzaileetan.

3.6.4.2. Helmholtzen ekuazio parametrikoa: $-50 < \alpha < -30$

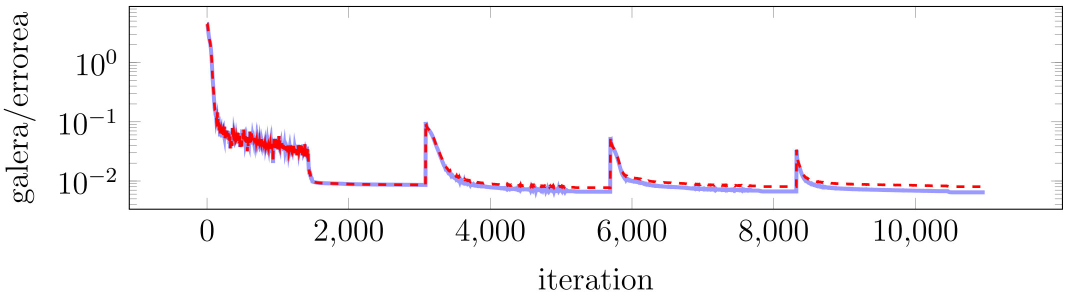
Kasu honen soluzio analitikoa honako hau da: $u^*(x) = C\sin(\sqrt{\alpha}x)$, $C = \frac{2\pi}{\sqrt{\alpha}\cos(\sqrt{\alpha})}$ izanik. Ehun lagineko datu-base bat aukeratuko dugu NNa entrenatzeko, eta honako probarako datu-base hau hautatuko dugu: $\{-50, -45, -40, -35, -30\}$. H^1-norma aukeratuko dugu galera-funtzioarentzako eta goian erabili dugun blokeka entrenagarria zen arkitektura bera. NNaren entrenamendua goiko bloke tamainen progresioa erabilita egiten badugu (ikus 3.21a. irudia), ohartuko gara oraingo honetan galerak ez duela errorearen H^1-norma gutxitzen. Alderantzizko matrizeak aurrebaldintzatzaile gisa erabiltzen baditugu (ikus 3.21b. irudia), emaitzak onargarriak dira. 3.22. irudian urratsez urratseko iragarpenak aurkezten dira, entrenamendua 3.21b. irudiaren arabera egiten denean.

Kontuan izan baliokideak direla alderantzizkoak erabiltzea eta honako galera-funtzio hau aplikatzea:

$$\mathcal{L}(\theta, D) = \frac{1}{|D|}\sum_{\alpha\in D}\Vert\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{u}_{\mathrm{FEM}}(\alpha)\Vert_{\mathbf{K}+\mathbf{M}}, \tag{3.33}$$

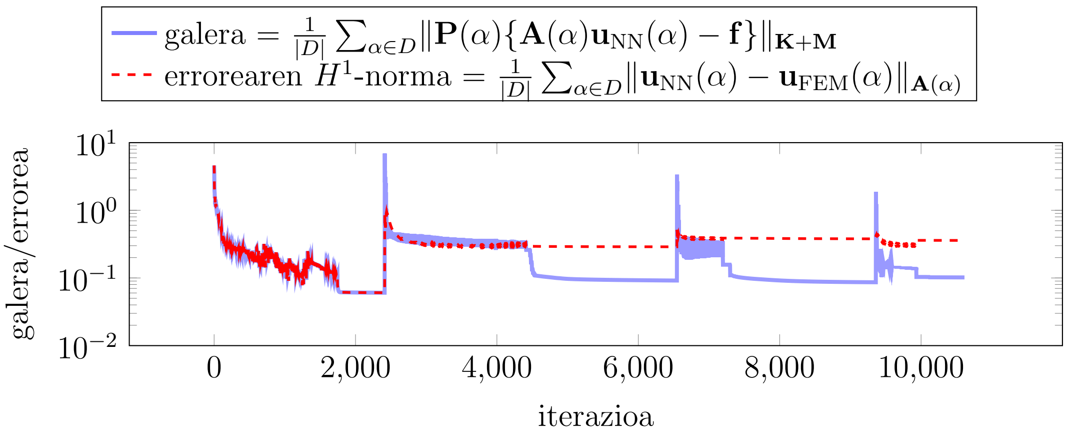


(a) Zortzi tamainako aurrebaldintzatzaileak erabilita.

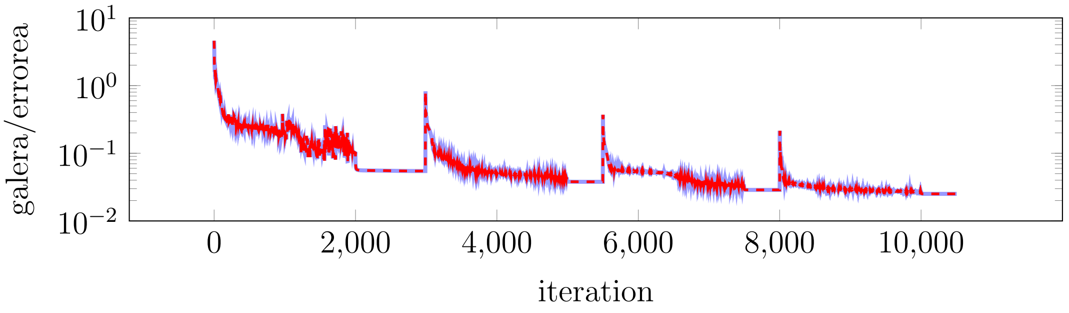


(b) 1., 2., 3., eta 4. urratsetan 8, 8, 16 eta 32 tamainako blokeak erabilita, hurrenez hurren.

3.20. irudia: Galera-funtzioaren eta errorearen energia-normaren eboluzioak lau urratsetan (3.32). probleman, $0 \leq \alpha \leq 200$ denean.

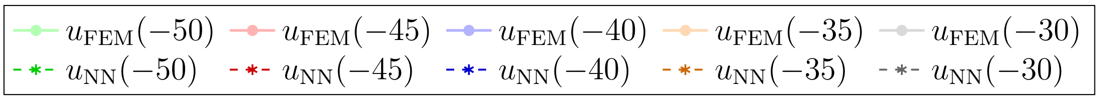


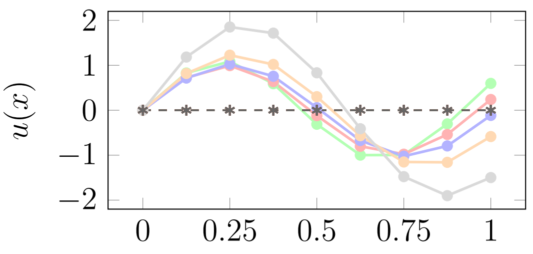
(a) 1., 2., 3., eta 4. urratsetan 8, 8, 16 eta 32 tamainako blokeak erabilita, hurrenez hurren.



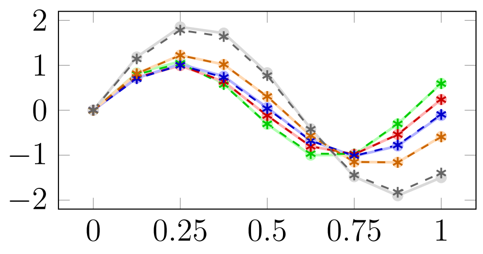
(b) Matrizeen alderantzizkoak erabilita aurrebaldintzatzaile gisa.

3.21. irudia: Galera-funtzioaren eta errorearen energia-normaren eboluzioak lau urratsetan (3.32). probleman, $-50 < \alpha < -30$ denean.

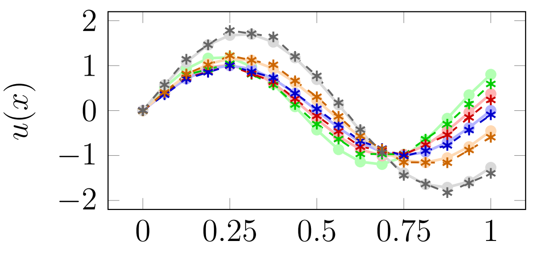




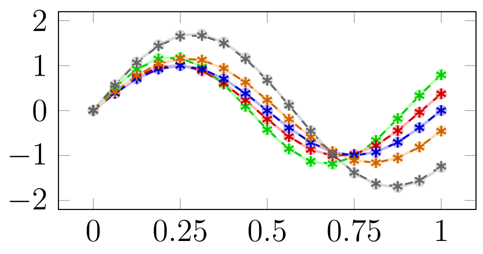
(a) $s = 1$ urratsa (entrenatu aurretik).



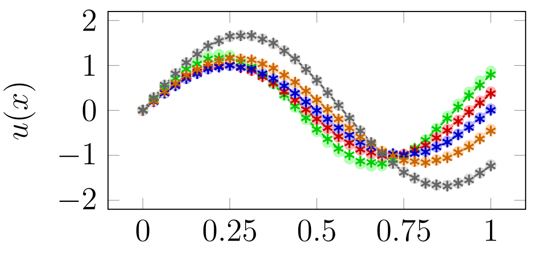
(b) $s = 1$ urratsa (entrenatu ondoren).



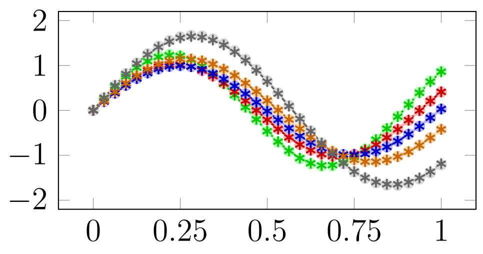
(c) $s = 2$ urratsa (entrenatu aurretik).



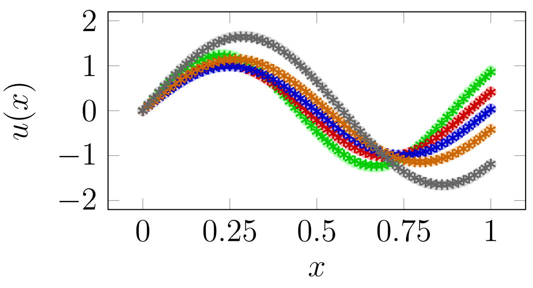
(d) $s = 2$ urratsa (entrenatu ondoren).



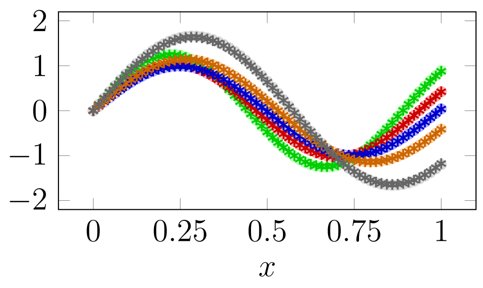
(e) $s = 3$ urratsa (entrenatu aurretik).



(f) $s = 3$ urratsa (entrenatu ondoren).



(g) $s = 4$ urratsa (entrenatu aurretik).



(h) $s = 4$ urratsa (entrenatu ondoren).

3.22. irudia: Lau urratsetan lortutako iragarpenak proba datu-basean (3.32). probleman, $-50 < \alpha < -30$ denean. Galera-funtzioan errorearen H^1-norma erabili dugu. $u_{\mathrm{FEM}}(\alpha)$ elementu finituetako soluzioa da eta $u_{\mathrm{NN}}(\alpha)$ DeepFEMaren iragarpena α parametroarentzako.

non $\mathbf{u}_{\mathrm{NN}}(\alpha)$ ikurrak DeepFEMaren iragarpen bektoriala adierazten baitu eta $\mathbf{u}_{\mathrm{FEM}}(\alpha)$ ikurrak FEMaren soluzio bektorea α parametroarentzako. Bektore horiek guztiak aurretiaz kalkulatuz, eta (3.33). ekuazioan dagoen galera-funtzioa erabiliz, entrenamendua alderantzizko matrizeak erabilita baino askoz merkeagoa izango litzateke. Sinpletasunagatik, kapitulu honetan zehar alderantzizkoak kalkulatu ditugu aurkeztutako arrazoiketarekin koherentzia mantentzeko, eta ez dugu exekuzio denborarik konparatu; ebazle eraginkor batek inoiz ez bailuke matrize baten alderantzizkoa esplizituki kalkulatuko [5, 169, 158].

4. The Deep Double Ritz Method

Summary. Residual minimization is a widely used technique for solving Partial Differential Equations in variational form. It minimizes the dual norm of the residual, which naturally yields a saddle-point (min–max) problem over the so-called trial and test spaces. In the context of Neural Networks, we can address this min–max approach by employing one Neural Network to seek the trial minimum while another Neural Network seeks the test maximizers. However, the resulting method is numerically unstable as we approach the trial solution. To overcome this, we reformulate the residual minimization as an equivalent minimization of a Ritz functional fed by optimal test functions computed from another Ritz functional minimization. We call the resulting scheme the *Deep Double Ritz Method*, which combines two Neural Networks for approximating trial and optimal test functions along a nested double Ritz minimization strategy. Numerical results on different diffusion and convection problems support the robustness of our method up to the approximation properties of the considered Neural Networks and the training capacity of the optimizers. *Refer to [217] for the published version.*

4.1. Introduction

Within the variational framework introduced in Section A.5.2/1.5.2, residual minimization reads as a saddle-point (min-max) problem as follows:

$$\min_{u\in\mathbb{U}} \max_{v\in\mathbb{V}\setminus\{0\}} \frac{\langle Bu-l,v\rangle_{\mathbb{V}'\times\mathbb{V}}}{\|v\|_{\mathbb{V}}}, \tag{4.1}$$

where $\mathbb{U}$ and $\mathbb{V}$ are the trial and test spaces, $B:\mathbb{U}\longrightarrow\mathbb{V}'$ is the differential operator governing the considered BVP in variational form, $\mathbb{V}'$ is the dual space of $\mathbb{V}$, and $l\in\mathbb{V}'$ is the right-hand side. In [232, 20], the authors proposed addressing this optimization scheme employing Generative Adversarial Networks (GANs) [78, 79] by approximating u and v with two NNs. Unfortunately, this approach presents a severe numerical limitation: the Lipschitz continuity constant of the test maximizers with respect to the trial functions might become arbitrarily large when approaching the exact solution. Moreover, the corresponding test maximizer is highly non-unique in the limit. In consequence, we end up with an inherent lack of numerical stability that is easily confirmed by numerical experiments with simple model problems.

To overcome the above limitations, we reformulate residual minimization as a minimization of a Ritz functional fed by optimal test functions [55, 58]. Since optimal test functions are generally unavailable, we compute them for each trial function using another Ritz method. Thus, the resulting scheme is a nested double-loop Ritz minimization method: the outer loop seeks the trial solution, while the inner loop seeks the optimal test function for each trial function. We call the resulting scheme the *Double Ritz Method*.

In some occasions, the trial-to-test operator that maps each trial function with the corresponding optimal test function is available. For example, when the problem is symmetric and positive-definite, and we consider the norm induced by the bilinear form for the trial and test spaces, the trial-to-test operator is the identity; or when selecting the strong variational formulation, the trial-to-test operator is the one given by the PDE operator. In these cases, the Double Ritz Method reduces to a single-loop Ritz minimization (this will be further discussed in Section 4.2.4). Thus, the Double Ritz Method is a general method for solving PDEs in different variational forms, which in some particular cases simplifies into a single-loop Ritz minimization method.

Thanks to NNs, we find a simple and advantageous computational framework to approximate and connect the trial and test functions between the inner- and outer-loop minimizations in the Double Ritz Method, a task that is challenging to tackle with traditional numerical methods. Herein, we propose using one NN to represent the trial functions and another NN to represent the local actions of the trial-to-test operator. Thus, the composition of both NNs represents the (optimal) test functions, and we preserve the trial dependence of the test functions during the entire process. We call the resulting NN-based method the *Deep Double Ritz Method (D^{2}RM)*.

While the D^{2}RM replicates existing residual minimization methods in the context of NNs, we fall short of providing a detailed mathematical convergence analysis due to the manifold structure of NNs that departs from the traditional vector-space-based mathematical approach (recall Chapter 2). Related to this, we encounter the usual drawbacks of lack of convexity between the objective/loss function with respect to the trainable parameters, which prevents us from making a proper diagnosis of the optimizer during training.

The remainder of this chapter is organized as follows. Section 4.2 formalizes the variational setting introduced in Section A.5.2 and derives the Double Ritz Method at the continuum level. Section 4.3 introduces the D^{2}RM within the NN framework. Section 4.4 provides implementation details of the addressed methods and Section 4.5 develops on numerical experimentation.

4.2. From residual to Ritz minimizations

We introduce the residual minimization framework, followed by a saddle-point reformulation and an alternative Double Ritz scheme at the continuum level. Subsequently, we describe three particular cases for which the Double Ritz method simplifies into a single Ritz method.

4.2.1. Residual minimization

Let

$$\left\{ \begin{array}{l} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ b(u^*, v) = l(v), \ \forall v \in \mathbb{V}, \end{array} \right. \tag{4.2a}$$

where $\mathbb{U}$ and $\mathbb{V}$ are real Hilbert trial and test spaces, respectively, $b : \mathbb{U} \times \mathbb{V} \longrightarrow \mathbb{R}$ is a bilinear form, and $l : \mathbb{V} \longrightarrow \mathbb{R}$ is a continuous linear functional. Equivalently, in operator form:

$$\left\{ \begin{array}{l} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ Bu^* = l, \end{array} \right. \tag{4.2b}$$

where $B : \mathbb{U} \longrightarrow \mathbb{V}'$ is the operator defined by $\langle Bu, v \rangle_{\mathbb{V}' \times \mathbb{V}} := b(u, v)$, $\mathbb{V}'$ denotes the topological dual of $\mathbb{V}$, and $l \in \mathbb{V}'$. The equivalent residual minimization formulation reads as

$$u^* = \arg \min_{u \in \mathbb{U}} \|Bu - l\|_{\mathbb{V}'}, \tag{4.3}$$

where $Bu - l \in \mathbb{V}'$ is the *residual* for each trial function $u \in \mathbb{U}$.

To guarantee well-posedness of (4.2)–(4.3), we assume the hypotheses of the Babuška–Lax–Milgram Theorem [17], which according to our presentation translate into that B is an isomorphism that is bounded from above and below, i.e., there exist some positive constants $\gamma \leq M$ such that it satisfies

$$\gamma \|u\|_{\mathbb{U}} \leq \|Bu\|_{\mathbb{V}'} \leq M \|u\|_{\mathbb{U}}, \qquad u \in \mathbb{U}. \tag{4.4}$$

Then, the error in $\mathbb{U}$ is equivalent to the residual in $\mathbb{V}'$ in the following sense:

$$\frac{1}{M} \|Bu - l\|_{\mathbb{V}'} \leq \|u - u^*\|_{\mathbb{U}} \leq \frac{1}{\gamma} \|Bu - l\|_{\mathbb{V}'}, \qquad u \in \mathbb{U}. \tag{4.5}$$

Below, we examine two alternatives to evaluate and minimize the residual in its dual norm.

4.2.2. Saddle-point problem (min-max optimization)

The norm in $\mathbb{V}'$ is defined in terms of the norm in $\mathbb{V}$ as

$$\|t\|_{\mathbb{V}'} := \sup_{v\in\mathbb{V}\setminus\{0\}} \frac{\langle t, v\rangle_{\mathbb{V}}}{\|v\|_{\mathbb{V}}} = \sup_{v\in\mathbb{V}\setminus\{0\}} \left\langle t, \frac{v}{\|v\|_{\mathbb{V}}}\right\rangle_{\mathbb{V}'\times\mathbb{V}}, \qquad t \in \mathbb{V}'. \tag{4.6}$$

Hence, combining (4.3) and (4.6) yields

$$u^* = \arg\min_{u\in\mathbb{U}} \max_{v\in\mathbb{V}\setminus\{0\}} \mathcal{F}_{\min}^{\max}(u, v) \tag{4.7a}$$

with

$$\mathcal{F}_{\min}^{\max}(u, v) := \left\langle Bu - l, \frac{v}{\|v\|_{\mathbb{V}}}\right\rangle_{\mathbb{V}'\times\mathbb{V}} = b\left(u, \frac{v}{\|v\|_{\mathbb{V}}}\right) - l\left(\frac{v}{\|v\|_{\mathbb{V}}}\right). \tag{4.7b}$$

When $\mathcal{F}_{\min}^{\max}$ is fed by the exact solution u^*, the resulting operator becomes the null functional, i.e., $\mathcal{F}_{\min}^{\max}(u^*, \cdot) = 0 \in \mathbb{V}'$, which implies a highly non-uniqueness of the test maximizer in the limit—indeed, any element in $\mathbb{V}\setminus\{0\}$ is a test maximizer for u^*. Outside this singular case, the operator that maps each trial function $u \in \mathbb{U}$ to its unitary test maximizer is well-defined but not Lipschitz continuous when approaching the exact solution u^*, leading to an unstable numerical method. We formalize this in the following two items:

- Let $v_{\max} : \mathbb{U} \setminus \{u^*\} \longrightarrow \mathbb{V}$ be the mapping that for each trial function returns the test maximizer of the actions of the residual $Bu - l \in \mathbb{V}'$ over the unitary sphere, i.e.,

  $$v_{\max}(u) := \arg\max_{\|v\|_{\mathbb{V}}=1} \langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}. \tag{4.8}$$

  Then, $v_{\max}(u) \in \mathbb{V}$ is unique for each $u \in \mathbb{U} \setminus \{u^*\}$.

  Proof. Let $u \in \mathbb{U}\setminus\{u^*\}$. By the Riesz Representation Theorem, there exists a unique $r_u \in \mathbb{V}$ such that

  $$(r_u, v)_{\mathbb{V}} = \langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}, \qquad \forall v \in \mathbb{V}, \tag{4.9}$$

  and

  $$\|r_u\|_{\mathbb{V}} = \|Bu - l\|_{\mathbb{V}'} = \max_{\|v\|_{\mathbb{V}}=1} \langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}. \tag{4.10}$$

  Let $v_{\max} = v_{\max}(u)$ be a test maximizer of (4.10) in the unitary sphere of $\mathbb{V}$, i.e.,

  $$\langle Bu - l, v_{\max}\rangle_{\mathbb{V}'\times\mathbb{V}} = \max_{\|v\|_{\mathbb{V}}=1} \langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}. \tag{4.11}$$

By the Cauchy-Schwarz inequality:

$$\begin{aligned} 0 < \|r_u\|_{\mathbb{V}} = \|Bu-l\|_{\mathbb{V}'} = \langle Bu-l, v_{\max}\rangle_{\mathbb{V}'\times\mathbb{V}} = (r_u, v_{\max})_{\mathbb{V}} && (4.12a)\\ \leq \|r_u\|_{\mathbb{V}}\, \|v_{\max}\|_{\mathbb{V}}, && (4.12b)\end{aligned}$$

where the equality holds if and only if $v_{\max} = \lambda r_u$ for some $\lambda > 0$. Then, $v_{\max} = \frac{r_u}{\|r_u\|_{\mathbb{V}}}$ is unique. □

- $v_{\max}$ is not Lipschitz continuous around any (reduced) neighborhood of u^*, i.e., there does not exist a constant $0 < C < \infty$ such that

$$\|v_{\max}(u_1) - v_{\max}(u_2)\|_{\mathbb{V}} \leq C\|u_1-u_2\|_{\mathbb{U}}, \qquad \forall u_1, u_2 \in \mathbb{U}\setminus\{u^*\}. \tag{4.13}$$

Proof. Assume by contradiction that there exists $0 < C < \infty$ such that (4.13) holds, and let $u_2 = 2u^* - u_1$ with $u_1 \neq u^*$. Then,

$$2 = \|v_{\max}(u_1) - v_{\max}(u_2)\|_{\mathbb{V}} \leq C\|u_1-u_2\|_{\mathbb{U}} = 2C\|u_1-u^*\|_{\mathbb{U}}. \tag{4.14}$$

Letting $u_1 \to u^*$, we obtain $C \to \infty$. □

4.2.3. Double Ritz Method with optimal test functions

The Riesz Representation Theorem allows us to work isometrically in the test space instead of in its dual. In particular, for the residual, we have

$$\|Bu-l\|_{\mathbb{V}'} = \|\mathfrak{R}_{\mathbb{V}}^{-1}(Bu-l)\|_{\mathbb{V}}, \qquad u \in \mathbb{U}, \tag{4.15}$$

where $\mathfrak{R}_{\mathbb{V}} : \mathbb{V} \ni v \longmapsto (v,\cdot)_{\mathbb{V}} \in \mathbb{V}'$ denotes the Riesz operator. This relation suggests considering the *trial-to-test* operator $T : \mathbb{U} \longrightarrow \mathbb{V}$ defined by $T := \mathfrak{R}_{\mathbb{V}}^{-1}B$ [58] since it relates the error in $\mathbb{U}$ with the Riesz representative of the residual in $\mathbb{V}$,

$$T(u-u^*) = \mathfrak{R}_{\mathbb{V}}^{-1}(Bu-l) \in \mathbb{V}, \qquad u \in \mathbb{U}. \tag{4.16}$$

The images of trial functions through T are known as *optimal test functions* [55], and they allow us to rewrite (4.2) in terms of the following symmetric and positive-definite variational problem:

$$\begin{cases} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ (Tu^*, Tu)_{\mathbb{V}} = l(Tu), \ \forall u \in \mathbb{U}, \end{cases} \tag{4.17}$$

Proof. On the one hand, $b(u,v) = \langle Bu, v\rangle_{\mathbb{V}'\times\mathbb{V}} = (Tu, v)_{\mathbb{V}}$ for all $u \in \mathbb{U}$ and all $v \in \mathbb{V}$. On the other hand, T is an isomorphism because so is B. Hence, testing with all $v \in \mathbb{V}$ is equivalent to testing with $Tu \in \mathbb{V}$ for all $u \in \mathbb{U}$. □

Because the bilinear form $(T\cdot, T\cdot) : \mathbb{U} \times \mathbb{U} \longrightarrow \mathbb{R}$ of (4.17) is symmetric and positive definite, we can reformulate (4.17) in terms of a quadratic-functional minimization combining the "usual" Ritz functional and the trial-to-test operator as follows:

$$u^* = \arg\min_{u \in \mathbb{U}} \mathcal{F}_T(u), \tag{4.18a}$$

with

$$\mathcal{F}_T(u) := \frac{1}{2}\|Tu\|_{\mathbb{V}}^2 - l(Tu). \tag{4.18b}$$

One might interpret that $\mathcal{F}_T$ acts as a generalization of the "usual" Ritz functional[1] $\mathcal{F}(\cdot) = \frac{1}{2}\|\cdot\|_{\mathbb{V}}^2 - l(\cdot)$ into problems that are not necessarily symmetric or positive-definite. The relation between residual minimization (4.3) and our proposed generalized Ritz minimization (4.18) is

$$\mathcal{F}_T(u) - \mathcal{F}_T(u^*) = \frac{1}{2}\|Bu - l\|_{\mathbb{V}'}^2. \tag{4.19}$$

Proof. According to (4.17), we can write $\mathcal{F}_T(u) = \frac{1}{2}(Tu, Tu)_{\mathbb{V}} - (Tu^*, Tu)$ for all $u \in \mathbb{U}$. Straightforward calculations lead to the desired identity. □

Now, the challenge is to compute $Tu \in \mathbb{V}$ when iterating along $u \in \mathbb{U}$. Because finding $Tu \in \mathbb{V}$ is equivalent to solving the symmetric and positive-definite variational problem

$$\begin{cases} \text{Find } Tu \in \mathbb{V} \text{ such that} \\ (Tu, v)_{\mathbb{V}} = b(u, v), \ \forall v \in \mathbb{V}, \end{cases} \tag{4.20}$$

following the same arguing as before, we can reformulate it as a corresponding Ritz minimization for each given/fixed $u \in \mathbb{U}$ as follows:

$$Tu = \arg\min_{v \in \mathbb{V}} \mathcal{F}_u^{\text{opt}}(v), \tag{4.21a}$$

with

$$\mathcal{F}_u^{\text{opt}}(v) := \frac{1}{2}\|v\|_{\mathbb{V}}^2 - b(u, v). \tag{4.21b}$$

However, characterizing Tu by means of (4.21a) remains impractical when iterating along $u \in \mathbb{U}$ to minimize (4.18) in the absence of knowledge of T[2]. To overcome this, instead of iterating along $\mathbb{V}$ to seek the optimal test function for

[1]The Ritz method [185] is typically reserved for symmetric and positive-definite problems due to the natural "energy" concept arising from those kinds of problems. In this way, the Ritz functional is typically presented as $\mathcal{F}(\cdot) = \frac{1}{2}b(\cdot,\cdot) - l(\cdot)$, where the inner product of the underlying Hilbert space is selected as the bilinear form, i.e., $(\cdot,\cdot)_{\mathbb{V}} = b(\cdot,\cdot)$.

[2]Intuitively, the minimizer in (4.21a) is delivered with an *implicit* dependence on $u \in \mathbb{U}$. Consequently, trying to iterate along $\mathbb{U}$ via (4.18) becomes challenging due to the absence of explicit dependencies to deal with.

a given $u \in \mathbb{U}$, we consider seeking along a "convenient" family $\mathbb{M}$ of operators from $\mathbb{U}$ to $\mathbb{V}$[3]. Then, we reformulate (4.21a) as seeking a candidate $\tau_u \in \mathbb{M}$ that when acting on $u \in \mathbb{U}$ delivers $Tu \in \mathbb{V}$, namely, given $u \in \mathbb{U}$,

$$\tau_u = \arg\min_{\tau \in \mathbb{M}} \mathcal{F}_u^{\text{opt}}(\tau(u)). \tag{4.22}$$

This provides a framework where a minimizer τ_u of (4.22) acts as T *only* at the given $u \in \mathbb{U}$, i.e., $\tau_u(u) = Tu$ but $\tau_u(w)$ might differ from Tw whenever $u \neq w$. Moreover, depending on the construction of $\mathbb{M}$, τ_u is possibly non-unique.

Hence, by performing a nested minimization of (4.22) within (4.18), we solve the problem at hand without explicitly dealing with the full operator T. We call this the *Double Ritz Method*. Algorithm 2 depicts its nested-loop optimization strategy that iterates separately either with elements in $\mathbb{U}$ or in $\mathbb{M}$.

Algorithm 2: Double Ritz Method

```
Initialize $u \in \mathbb{U}$ and $\tau \in \mathbb{M}$;
while not converged do
    Find $\tau_u \in \arg\min_{\tau \in \mathbb{M}} \mathcal{F}_u^{\text{opt}}(\tau(u))$;
    $u$ = following candidate in $\mathbb{U}$ minimizing $\mathcal{F}_{\tau_u}(u)$;
return $u$
```

4.2.4. Generalized Ritz Methods

We show three scenarios where the Double Ritz Method simplifies into a single Ritz minimization.

(a) **(Traditional) Ritz Method.** If the bilinear form $b(\cdot,\cdot)$ is symmetric and positive definite, we have $\mathbb{V} = \mathbb{U}$. Considering the inner product as the bilinear form yields $Tu = u$ for all $u \in \mathbb{U}$, i.e., T is the identity operator.

(b) **Strong formulation.** Let $A : \mathcal{D}(A) \longrightarrow L^2(\Omega)$ denote a PDE operator with domain $\mathcal{D}(A)$. Considering $b(u,v) = (Au, v)_{L^2(\Omega)}$, $\mathbb{U} = \mathcal{D}(A)$ equipped with the graph norm, and $\mathbb{V} = L^2(\Omega)$, we obtain $Tu = Au$ for all $u \in \mathbb{U}$, i.e., T is the PDE operator.

(c) **Ultraweak formulation.** Let $A' : \mathcal{D}(A') \longrightarrow L^2(\Omega)$ denote the adjoint operator of the PDE operator $A : \mathcal{D}(A) \longrightarrow L^2(\Omega)$, i.e., A' satisfies

[3]At a continuum level, we should ensure that $\mathbb{M}$ is able to produce optimal test functions for any "iterable" trial function, e.g., by assuming that there exists a family of operators L_u in $\mathbb{M}$ such that $Tu \in \text{Im}(L_u)$ for all "iterable" $u \in \mathbb{U}$ (note that this does not imply $T \in \mathbb{M}$).

$(Au,v)_{L^2(\Omega)} = (u,A'v)_{L^2(\Omega)}$ for all $u \in \mathcal{D}(A)$ and $v \in \mathcal{D}(A')$. Considering $b(u,v) = (u,A'v)_{L^2(\Omega)}$, $\mathbb{U} = L^2(\Omega)$, and $\mathbb{V} = \mathcal{D}(A')$ equipped with the graph norm, we obtain $A'Tu = u$ for all $u \in \mathbb{U}$.

Following the ideas of [41, 56], we can first solve for the optimal test function of the trial solution:

$$Tu^* = \arg\min_{v\in\mathbb{V}} \mathcal{F}'(v), \qquad \mathcal{F}'(v) := \frac{1}{2}\|A'v\|^2_{L^2(\Omega)} - l(v), \tag{4.23}$$

and then apply A' to the minimizer to recover the trial solution: $u^* = A'Tu^*$. We call *Adjoint Ritz Method* to this Ritz minimization with post-processing based on the use of the A' operator.

To illustrate the above three cases, we show a simple PDE problem with different variational formulations.

Example 4.1 (Poisson's Equation)**.** Let $f \in L^2(0,1)$ and consider the following 1D pure diffusion equation with homogeneous Dirichlet boundary conditions over $\Omega = (0,1)$:

$$\begin{cases} -u'' = f, \\ u(0) = u(1) = 0. \end{cases} \tag{4.24}$$

We multiply the PDE by a test function and integrate over Ω. Depending on the number of times we integrate by parts to derive the variational formulation, we obtain the following scenarios:

1. **Strong formulation.** Without integration by parts. Then, $\mathbb{U} = H^2(0,1)\cap H^1_0(0,1)$, $\mathbb{V} = L^2(0,1)$, $b(u,v) = \int_0^1 -u''v$, and $Tu = -u''$ for all $u \in \mathbb{U}$. This is the case (b) above.

2. **Weak formulation.** We integrate by parts once, passing one derivative from the trial to the test function. Then, $\mathbb{U} = H^1_0(0,1) = \mathbb{V}$, $b(u,v) = \int_0^1 u'v'$, $(v_1,v_2)_{\mathbb{V}} = b(v_1,v_2)$, and $Tu = u$ for all $u \in \mathbb{U}$. This is the case (a) above.

3. **Ultraweak formulation.** We integrate by parts twice, passing the two derivatives of the trial to the test function. Then, $\mathbb{U} = L^2(0,1)$, $\mathbb{V} = H^2(0,1) \cap H^1_0(0,1)$, $(v_1,v_2)_{\mathbb{V}} = (-v_1'', -v_2'')_{L^2(\Omega)}$, $b(u,v) = \int_0^1 -uv''$, and $-(Tu)'' = u$ for all $u \in \mathbb{U}$. This is the case (c) above.

All three variational formulations are valid, although they exhibit different convergence behaviors that go beyond the scope of this work.

4.3. Approximation with Neural Networks

In practice, instead of seeking along $\mathbb{U}$, $\mathbb{V}$, and $\mathbb{M}$, we seek along corresponding computationally accessible subsets generated by NNs. This section elaborates on the NN setting for the computability of the proposed methods following the presentation and notation of Chapter 2.

4.3.1. Parameterization

Let u_{NN}, v_{NN} and τ_{NN} be FFNN architectures with corresponding sets of learnable parameters θ_u, θ_v and θ_τ and sets of realizations $\mathbb{U}_{\text{NN}}$, $\mathbb{V}_{\text{NN}}$, and $\mathbb{M}_{\text{NN}}$, respectively.

To ensure the containment of the set of realizations within the trial space (i.e., $\mathbb{U}_{\text{NN}} \subset \mathbb{U}$), we select smooth activation functions (e.g., tanh) and strongly impose Dirichlet boundary conditions by considering $u^{\xi}_{\text{NN}} : \Omega \longrightarrow \mathbb{R}$ defined by

$$u^{\xi}_{\text{NN}}(x) := \xi(x) u_{\text{NN}}(x), \qquad x \in \Omega, \tag{4.25}$$

where $u_{\text{NN}} : \Omega \longrightarrow \mathbb{R}$ is the previously considered boundary-free FFNN architecture and $\xi : \Omega \longrightarrow \mathbb{R}$ is a non-trainable smooth cut-off function that vanishes only on the Dirichlet boundary. For simplicity, we drop the superscript ξ notation, meaning that u_{NN} will directly refer to u^{ξ}_{NN} from now on. We apply similar criteria to v_{NN} and τ_{NN}.

Table 4.1 summarizes some of the methods introduced in Section 4.2 within the framework of NNs.

Method	Objective function(s)	Optimization
Weak Adversarial Networks	$\mathcal{F}^{\max}_{\min}(u_{\text{NN}}, v_{\text{NN}}) = b\left(u_{\text{NN}}, \frac{v_{\text{NN}}}{\\|v_{\text{NN}}\\|_{\mathbb{V}}}\right) - l\left(\frac{v_{\text{NN}}}{\\|v_{\text{NN}}\\|_{\mathbb{V}}}\right)$	$u^*_{\text{NN}} = \arg \min_{u_{\text{NN}} \in \mathbb{U}_{\text{NN}}} \max_{v_{\text{NN}} \in \mathbb{V}_{\text{NN}}} \mathcal{F}^{\max}_{\min}(u_{\text{NN}}, v_{\text{NN}})$
Generalized Deep Ritz Method	$\mathcal{F}_T(u_{\text{NN}}) = \frac{1}{2}\\|T u_{\text{NN}}\\|^2_V - l(T u_{\text{NN}})$ T is the “available” trial-to-test operator	$u^*_{\text{NN}} = \arg \min_{u_{\text{NN}} \in \mathbb{U}_{\text{NN}}} \mathcal{F}_T(u_{\text{NN}})$
Deep Double Ritz Method	$\mathcal{F}_{\tau_{\text{NN}}}(u_{\text{NN}}) = \frac{1}{2}\\|\tau_{\text{NN}}(u_{\text{NN}})\\|^2_V - l(\tau_{\text{NN}}(u_{\text{NN}}))$ $\mathcal{F}^{\text{opt}}_{u_{\text{NN}}}(\tau_{\text{NN}}(u_{\text{NN}})) = \frac{1}{2}\\|\tau_{\text{NN}}(u_{\text{NN}})\\|^2_V - b(u_{\text{NN}}, \tau_{\text{NN}}(u_{\text{NN}}))$	$u^*_{\text{NN}} = \arg \min_{u_{\text{NN}} \in \mathbb{U}_{\text{NN}}} \mathcal{F}_{\tau_{u_{\text{NN}}}}(u_{\text{NN}})$ $\tau_{u_{\text{NN}}} = \arg \min_{\tau_{\text{NN}} \in \mathbb{M}_{\text{NN}}} \mathcal{F}^{\text{opt}}_{u_{\text{NN}}}(\tau_{\text{NN}}(u_{\text{NN}}))$

Table 4.1: The saddle-point approach, the Generalized Ritz Method, and the Double Ritz Method in the context of NNs: *Weak Adversarial Networks (WANs)*, *the Generalized Deep Ritz Method (GDRM)*, and *the Deep Double Ritz Method (D^{2}RM)*. Here, the “arg” and “min/max” terms should be understood loosely according to the possible lack of existence of minimizers/maximizers discussed in Chapter 2.

4.3.2. Training

We approximate the corresponding integrals via quadrature rules, thus producing the loss functions (recall Chapter 2). We replace the calligraphic $\mathcal{F}$ with a calligraphic $\mathcal{L}$, maintaining the remaining symbology, to denote loss functions instead of objective functions, namely,

$$\mathcal{F}_{\min}^{\max}(u_{\text{NN}}, v_{\text{NN}}) \approx \mathcal{L}_{\min}^{\max}(\theta_u, \theta_v; \{x_j\}_{j=1}^N;), \tag{4.26a}$$

$$\mathcal{F}_T(u_{\text{NN}}) \approx \mathcal{L}_T(\theta_u; \{x_j\}_{j=1}^N), \tag{4.26b}$$

$$\mathcal{F}_{u_{\text{NN}}}^{\text{opt}}(\tau_{\text{NN}}(u_{\text{NN}})) \approx \mathcal{L}_{u_{\text{NN}}}^{\text{opt}}(\theta_\tau; \{x_j\}_{j=1}^N), \tag{4.26c}$$

where $\{x_j\}_{j=1}^N$ denotes the set of integration points employed during integral approximation (recall Section 2.2).

To maintain the advantages of stochastic integration, reduce the runtime and sample size—from thousands to hundreds of points, and control the integration error around singularities, we consider a *randomized (composite) intermediate-point quadrature rule* that generates random integration points following a beta $\beta(a, b)$ probability distribution [86, 225]. We conveniently tune the hyperparameters a and b according to our problem specifications. Algorithm 3 describes the randomized intermediate-point quadrature rule in $\Omega = (0, 1)$.

Algorithm 3: Randomized intermediate-point quadrature in $\Omega = (0, 1)$

```
Generate $x_i \in (0,1)$ for $1 \leq i \leq N$;
Sort $\{x_i : 1 \leq i \leq N\}$ so that $x_{i-1} < x_i$ for all $1 \leq i \leq N$;
Evaluate $I$ in $\{x_i : 1 \leq i \leq N\}$;
Define $m_0 := 0$, $m_i := (x_{i-1} + x_i)/2$ for $1 \leq i \leq N-1$, and $m_N := 1$;
Define $\omega_i := m_i - m_{i-1}$ for $1 \leq i \leq N$;
return $\sum_{i=1}^N \omega_i \cdot I(x_i)$, which approximates $\int_0^1 I(x)dx$;
```

Below, we detail the different optimization strategies when using NNs for the minimization schemes described in Section 4.2:

- **Weak Adversarial Networks (WANs).** We use two stochastic gradient-based optimizers [191] to readjust the learnable parameters during the min-max optimization. For simplicity, we illustrate the functioning with the classical Stochastic Gradient Descent (SGD) optimizer,

$$\theta_u = \theta_u - \lambda_u \frac{\partial \mathcal{L}_{\min}^{\max}}{\partial \theta_u}(\theta_u, \theta_v; \{x_j\}_{j=1}^N), \tag{4.27a}$$

and its maximization analogue, the Stochastic Gradient Ascent (SGA),

$$\theta_v = \theta_v + \lambda_v \frac{\partial \mathcal{L}_{\text{min}}^{\text{max}}}{\partial \theta_v}(\theta_u, \theta_v; \{x_j\}_{j=1}^N). \tag{4.27b}$$

Above, $\lambda_u, \lambda_v > 0$ denote the learning rates, and we dropped the iteration (sub)indexes for simplicity—recall (2.21). We perform multiple iterations on the ascent for each iteration on the descent (see Algorithm 4). We call *outer* and *inner loops* to the minimization and maximization processes, respectively, because of the nested optimization structure.

Algorithm 4: Training of Weak Adversarial Networks (WANs)

```
Initialize $\theta_u \in \Theta_u$ and $\theta_v \in \Theta_v$;
/* Outer-loop                                              */
while not converged do
    Randomly sample $\{x_j\}_{j=1}^N \subset \Omega$;
    $\theta_u = \theta_u - \lambda_u \frac{\partial \mathcal{L}_{\text{min}}^{\text{max}}}{\partial \theta_u}(\theta_u, \theta_v; \{x_j\}_{j=1}^N)$;
    /* Inner-loop                                          */
    while not converged do
        Randomly sample $\{x_j\}_{j=1}^N \subset \Omega$;
        $\theta_v = \theta_v + \lambda_v \frac{\partial \mathcal{L}_{\text{min}}^{\text{max}}}{\partial \theta_v}(\theta_u, \theta_v; \{x_j\}_{j=1}^N)$;
return $\theta_u$
```

- **Deep Double Ritz Method (D^{2}RM).** The training is similar to Algorithm 4, but modifying the loss when jumping from the outer to the inner loop, and performing only gradient-descents for minimizations (see Algorithm 5).

- **Generalized Deep Ritz Method (GDRM).** The optimization consists of a single-loop minimization (see Algorithm 6) when the trial-to-test operator T is "computably available". In particular, when the bilinear form is symmetric and positive definite (recall item (a) in Section 4.2.4), this method is the well-known Deep Ritz Method (DRM) [69].

- **Adjoint Deep Ritz Method (DRM)$'$.** The optimization consists of a single-loop minimization (as in Algorithm 6) with post-processing and is only valid in ultraweak formulations (recall item (c) in Section 4.2.4).

We consider the Adam optimizer [119] to carry out our experiments. In the case of nested optimizations, we select fully independent Adam optimizers for the inner

Algorithm 5: Training of the Deep Double Ritz Method (D^{2}RM)

```
Initialize $\theta_u \in \Theta_u$ and $\theta_\tau \in \Theta_\tau$;
/* Outer-loop                                        */
while not converged do
    Randomly sample $\{x_j\}_{j=1}^N \subset \Omega$;
    $\theta_u = \theta_u - \lambda_u \frac{\partial \mathcal{L}_{\tau_{\mathrm{NN}}}}{\partial \theta_u}(\theta_u, \theta_\tau; \{x_j\}_{j=1}^N)$;
    /* Inner-loop                                    */
    while not converged do
        Randomly sample $\{x_j\}_{j=1}^N \subset \Omega$;
        $\theta_\tau = \theta_\tau - \lambda_\tau \frac{\partial \mathcal{L}_{u_{\mathrm{NN}}}^{\mathrm{opt}}}{\partial \theta_\tau}(\theta_u, \theta_\tau; \{x_j\}_{j=1}^N)$;
return $\theta_u$
```

Algorithm 6: Training of the Generalized Deep Ritz Method (GDRM)

```
Initialize $\theta_u \in \Theta_u$;
while not converged do
    Randomly sample $\{x_j\}_{j=1}^N \subset \Omega$;
    $\theta_u = \theta_u - \lambda_u \frac{\partial \mathcal{L}_T}{\partial \theta_u}(\theta_u; \{x_j\}_{j=1}^N)$;
return $\theta_u$
```

and outer loops. We establish an accumulated maximum number of iterations for both nested loops, and we fix four inner-loop iterations for each outer-loop iteration[4] (as considered in [232, 20] for WANs[5]) unless otherwise specified.

[4]We made this decision motivated by the continuity of the elements involved during the process: a slight modification in θ_u translates into a slight variation in u_{NN} (continuity of the realization mapping—recall Section 2.1) that produces a slight variation in Tu_{NN} (continuity of T). Therefore, we expect (have the hope) that $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ can provide a good approximation of Tu_{NN} after a small number of iterations in θ_τ.

[5]The lack of Lipschitz continuity in the min-max approach suggests that this is a poor strategy for optimizing WANs.

4.4. Implementation

We use the TensorFlow 2 (TF2) library [3, 2] within Python to implement our neural network architectures and loss functions, and to manage the random creation and flow of data. Specifically, we accommodate all of our implementations to Keras (`tf.keras`).

4.4.1. Samples generation, input batch flow, encapsulation of models, and optimization

Our inputs to networks are samples over the domain Ω. After feeding our networks with the batch of inputs, we combine the batch of outputs in a single loss prediction. We encapsulate the networks and losses inside a main model whose input and output are the batch of samples and the loss prediction, respectively. From the loss prediction, we optimize the learnable parameters of the networks (i.e., we fit the learnable parameters to the data) with the Keras built-in Adam optimizer. Figure 4.1 illustrates the described process at each training iteration.

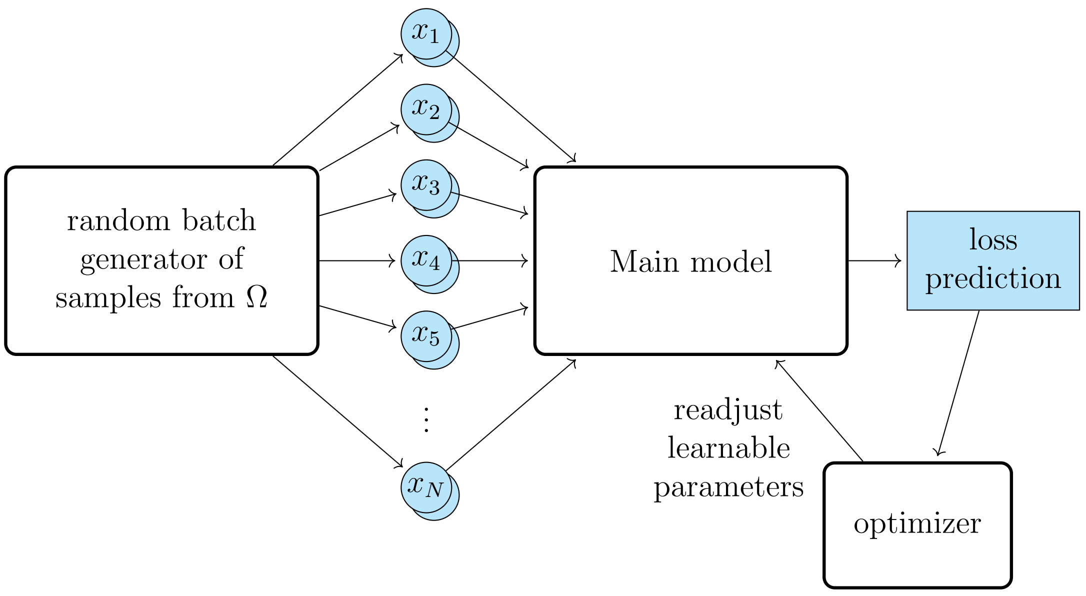

random batch generator of samples from Ω
x_1
x_2
x_3
x_4
x_5
$\vdots$
x_N
Main model
loss prediction
optimizer
readjust learnable parameters

Figure 4.1: Implementation sketch of the general flux for the proposed methods at each training iteration.

For the maximization involved in WANs, we reverse the sign of the gradients so that when feeding the (default) gradient-descent-based optimizer, it instead performs a gradient ascent.

4.4.2. Design of models by methods

We implement networks, losses, and operators as models and layers in TF2 from the redefinitions of the corresponding Keras base classes (`tf.keras.Model` and `tf.keras.Layer`). In WANs, we implement u_{NN} and v_{NN} as two independent models that are combined via a common non-trainable layer for the loss (see Figure 4.2). In the DRM, we implement u_{NN} as a model that subsequently connects with two non-trainable layers for the trial-to-test operator and the loss function, respectively (see Figure 4.3). In the D^{2}RM, we implement u_{NN} and τ_{NN} as two sequential models whose output feeds into two separate losses (see Figure 4.4). The loss functions are implemented as latent outputs of the main model, which, together with a TF2-suitable boolean variable, activate and deactivate alternatively. Despite the significant compilation time that the two-branch model (for the D^{2}RM) takes compared with one-branch models (for WANs and the DRM)[6], its fitting execution in graph mode is as fast as that of one-branch models.

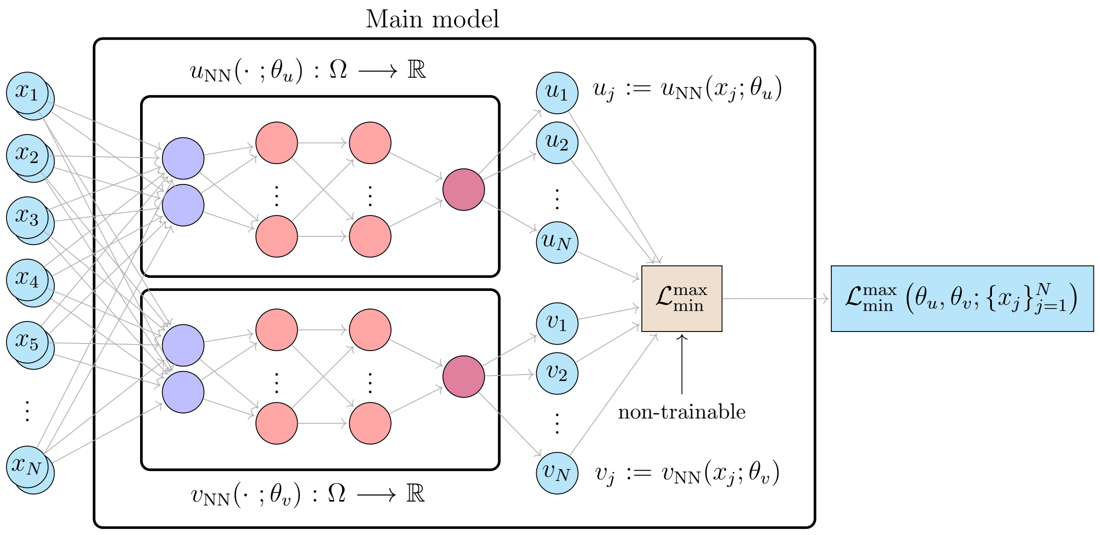

Main model
$u_{\text{NN}}(\cdot\,;\theta_u) : \Omega \longrightarrow \mathbb{R}$
$v_{\text{NN}}(\cdot\,;\theta_v) : \Omega \longrightarrow \mathbb{R}$
x_1
x_2
x_3
x_4
x_5
x_N
u_1
u_2
u_N
v_1
v_2
v_N
$u_j := u_{\text{NN}}(x_j;\theta_u)$
$v_j := v_{\text{NN}}(x_j;\theta_v)$
$\mathcal{L}^{\max}_{\min}$
non-trainable
$\mathcal{L}^{\max}_{\min}\left(\theta_u,\theta_v;\{x_j\}_{j=1}^N\right)$

Figure 4.2: Main model architecture for WANs. It consists of two independent NNs, u_{NN} and v_{NN}, combined via the loss function $\mathcal{L}^{\max}_{\min}$.

4.4.3. Graph-mode execution dynamics

We employ *callbacks* to avoid interrupting the graph execution mode carried out by the Keras fitting instruction (`.fit`). Callbacks act during fitting and enable

[6]Around one minute for the D^{2}RM vs. a couple of seconds for WANs and the DRM.

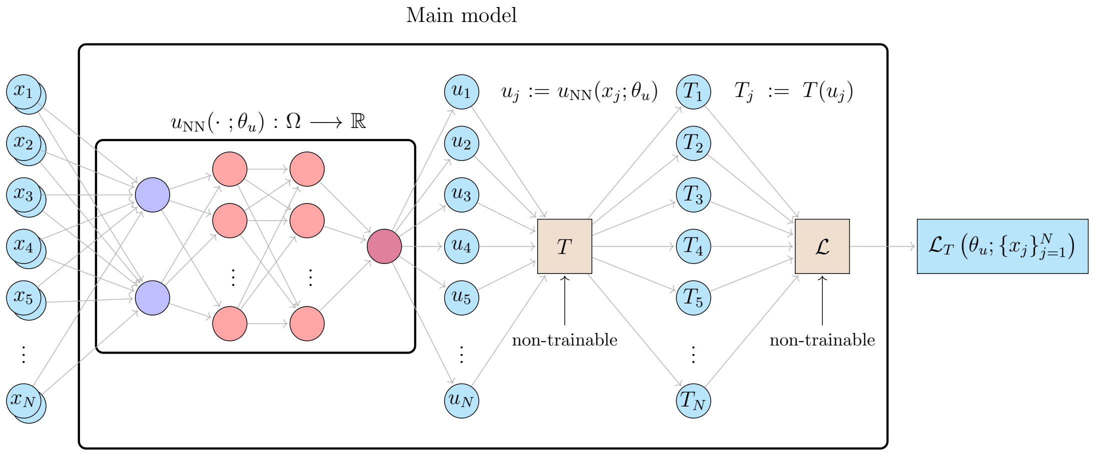

Main model
x_1
x_2
x_3
x_4
x_5
x_N
$u_{\text{NN}}(\cdot\,;\theta_u) : \Omega \longrightarrow \mathbb{R}$
u_1
u_2
u_3
u_4
u_5
u_N
$u_j := u_{\text{NN}}(x_j;\theta_u)$
T
non-trainable
T_1
T_2
T_3
T_4
T_5
T_N
$T_j := T(u_j)$
$\mathcal{L}$
non-trainable
$\mathcal{L}_T\left(\theta_u; \{x_j\}_{j=1}^N\right)$

Figure 4.3: Main model architecture for the DRM. It consists of a NN, u_{NN}, composed with the trial-to-test operator T and the loss function $\mathcal{L}$.

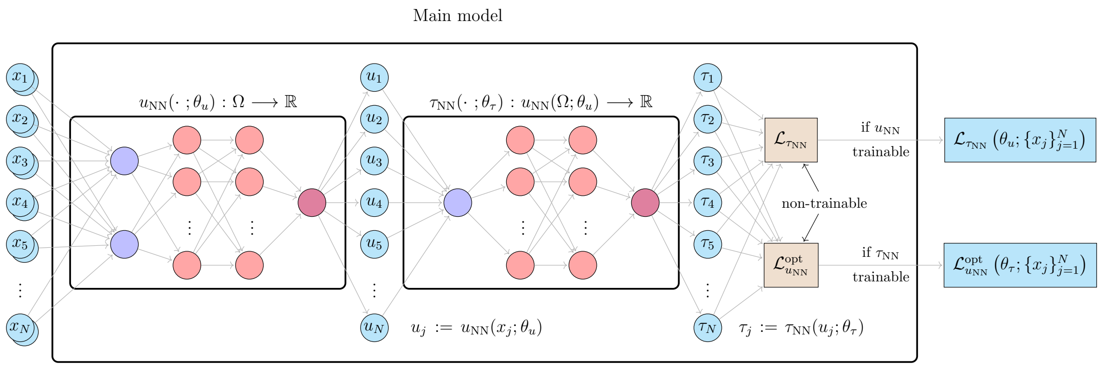

Main model
x_1
x_2
x_3
x_4
x_5
x_N
$u_{\text{NN}}(\cdot\,;\theta_u) : \Omega \longrightarrow \mathbb{R}$
u_1
u_2
u_3
u_4
u_5
u_N
$u_j := u_{\text{NN}}(x_j;\theta_u)$
$\tau_{\text{NN}}(\cdot\,;\theta_\tau) : u_{\text{NN}}(\Omega;\theta_u) \longrightarrow \mathbb{R}$
τ_1
τ_2
τ_3
τ_4
τ_5
τ_N
$\tau_j := \tau_{\text{NN}}(u_j;\theta_\tau)$
$\mathcal{L}_{\tau_{\text{NN}}}$
non-trainable
$\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}$
if u_{NN} trainable
if τ_{NN} trainable
$\mathcal{L}_{\tau_{\text{NN}}}\left(\theta_u; \{x_j\}_{j=1}^N\right)$
$\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}\left(\theta_\tau; \{x_j\}_{j=1}^N\right)$

Figure 4.4: Main model architecture for the D^{2}RM. It consists of two NNs, u_{NN} and τ_{NN}, equipped with the loss functions $\mathcal{L}_{\tau_{\text{NN}}}$ and $\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}$.

accessing certain elements of (main) models and modifying them. We utilize callbacks for loss monitoring (for WANs, the DRM, and the D^{2}RM), to activate or deactivate the trainability of the networks and switch between optimizers (for WANs and the D^{2}RM), or to interchange losses (for the D^{2}RM) during training when iterating either over the outer- or the inner-loop.

4.5. Numerical results

We show numerical experiments to compare the methods introduced above. Section 4.5.1 considers a simple model problem and compares WANs, the DRM, and the D^{2}RM. Section 4.5.2 makes a more profound comparison considering a parametric model problem. Section 4.5.3 and Section 4.5.4 consider sources that lead to singular problems in pure diffusion and convection equations, respectively. Finally, Section 4.5.5 considers a pure convection equation in 2D.

4.5.1. Initial comparison of WANs, the DRM, and the D^{2}RM on a simple problem

We select model problem (4.24) in weak form with source $f = -2$, so the exact solution is $u^* = x(x-1)$. Hence, $\mathbb{U} = H_0^1(0,1) = \mathbb{V}$ and T is the identity operator. We solve this problem by employing WANs, the DRM, and the D^{2}RM.

We select a two-layer fully-connected NN with 20 neurons on each layer and tanh activation functions for the architectures of u_{NN}, v_{NN}, and τ_{NN}. We perform 200 iterations for u_{NN} in the three methods: WANs, the DRM, and the D^{2}RM. Since in WANs and the D^{2}RM we established four iterations to approximate the test maximizers, we end up with a total of 1,000 training iterations (200 for the trial function, and 800 for the test functions). Moreover, we select batches of size 200 for the training and a uniform distribution for the sample generation.

Figure 4.5 shows the u_{NN} network predictions and errors of the three methods at the end of the training. We observe that WANs produce a larger error than the DRM and the D^{2}RM.

Figure 4.6 shows the loss evolution for WANs. Every five iterations, the loss decreases (minimization in u_{NN}) and increases in the remaining iterations (maximization in v_{NN} with fixed u_{NN}). From iteration 500 onwards, the loss stops improving and oscillates above the optimal value.

Figure 4.7 shows the evolution of the loss for the DRM. Here, we have a single minimization, so the observed noisy behavior of the loss towards the end of the training is attributed to the optimizer performance. Due to the lower complexity of the training, we achieve better convergence performance than with WANs.

Figure 4.8 shows the loss evolutions for the D^{2}RM. Here, we have a nested min-min optimization. At each iteration, we evaluate both $\mathcal{L}_{\tau_{\text{NN}}}$ and $\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}$, even if we are only optimizing with respect to one of them. We superimpose both losses, each one with its own scale (the left vertical axis corresponds to $\mathcal{L}_{\tau_{\text{NN}}}$ and the right vertical axis to $\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}$). Both losses exhibit a decreasing staircase shape with downward-sloping steps. Jumps occur when the optimization is performed with respect to $\mathcal{L}_{\tau_{\text{NN}}}$, which suggests that $\mathcal{L}^{\text{opt}}_{u_{\text{NN}}}$ takes longer to converge as it depends on the convergence of $\mathcal{L}_{\tau_{\text{NN}}}$ (inner- vs. outer-loop).

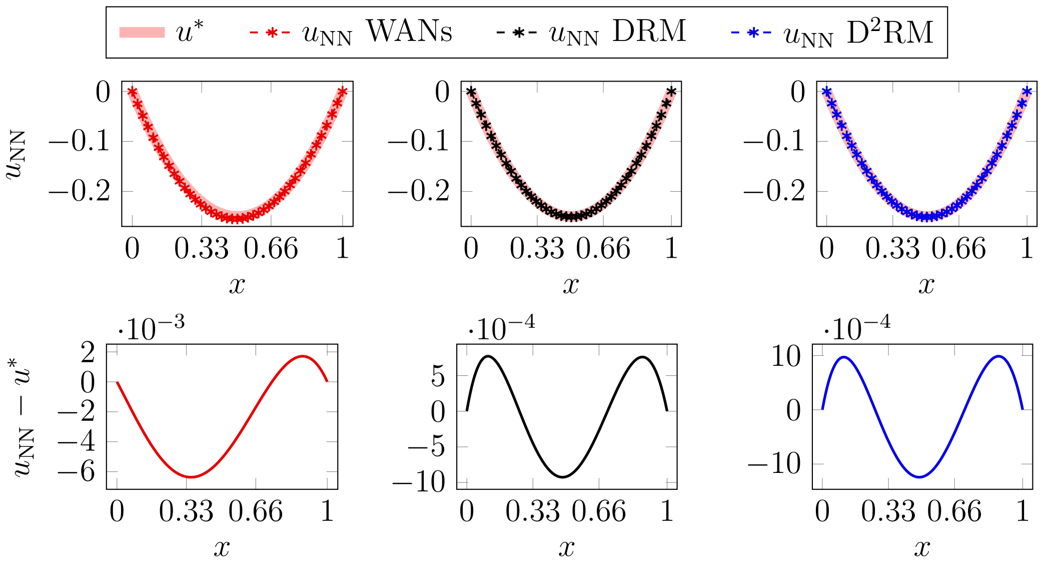


Figure 4.5: Trial network predictions and errors in WANs, the DRM, and the D^{2}RM at the end of the training in model problem (4.24) with exact solution $u^* = x(x-1)$.

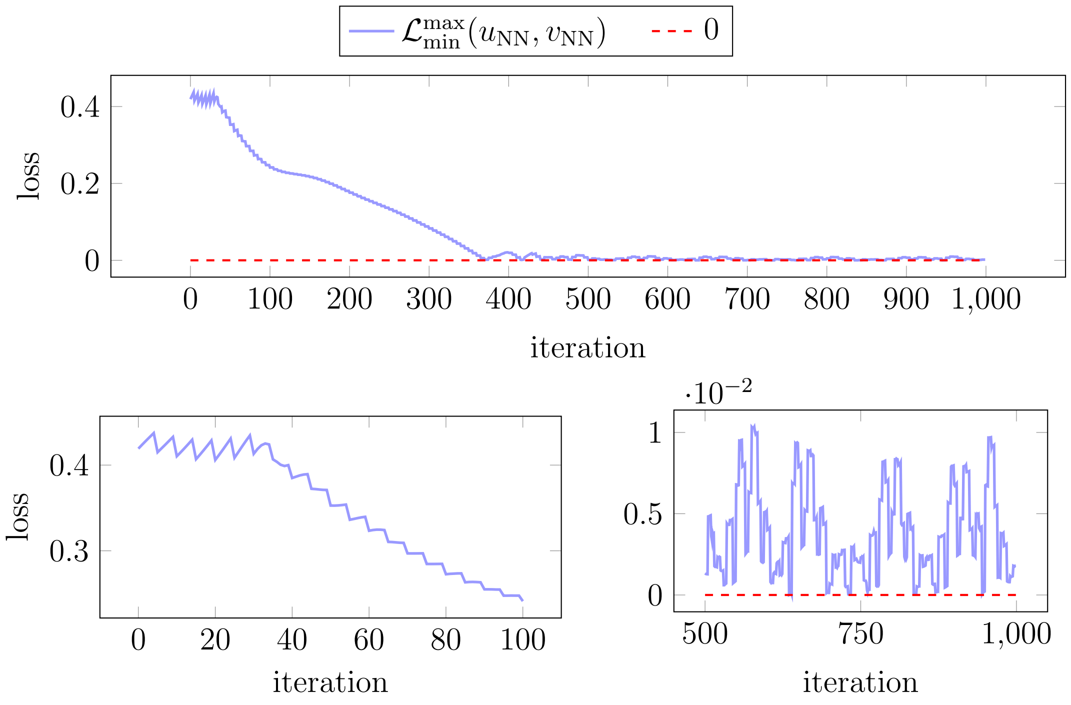


Figure 4.6: Loss evolution during the WANs training for model problem (4.24) with exact solution $u^* = x(x-1)$.

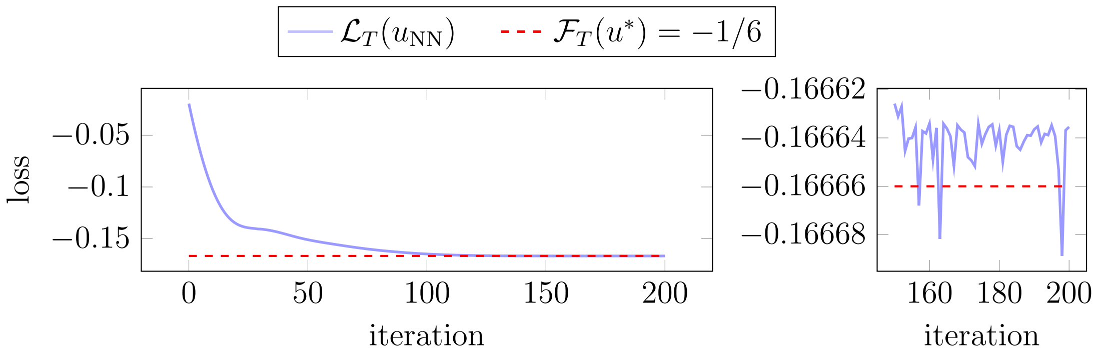


Figure 4.7: Loss evolution during the DRM training in model problem (4.24) with exact solution $u^* = x(x-1)$.

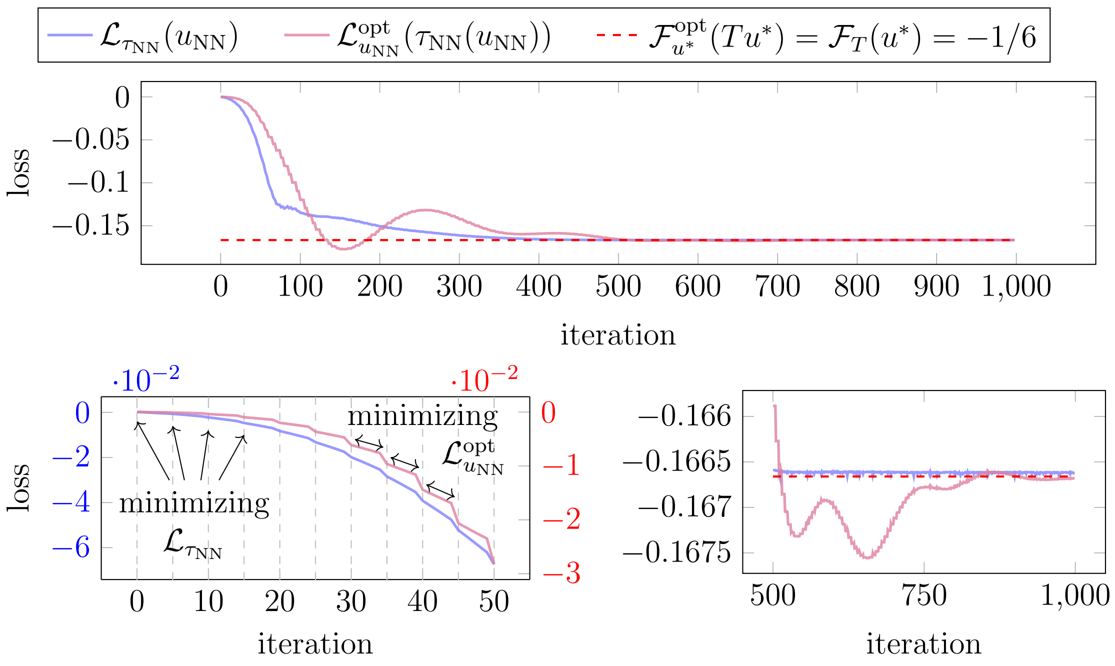


Figure 4.8: Loss evolution of the D^{2}RM training for model problem (4.24) with exact solution $u^* = x(x-1)$.

The relative errors of the trial network predictions[7] at the end of the training are 3.12%, 0.99%, and 1.31% in WANs, the DRM, and the D^{2}RM, respectively.

4.5.2. Comparison of WANs, the DRM, and the D^{2}RM on smooth problems

Now, we focus on the evolution of the relative error for the previous weak formulation of (4.24), but selecting the source so that the solution varies according to a parameter:

$$u^*_\alpha = x^\alpha(x-1) \in H^1_0(0,1), \qquad \alpha > 1/2. \tag{4.28}$$

4.5.2.1. Without singularities: $\alpha \geq 1$

We experiment individually for $\alpha \in \{2, 5, 10\}$ with 5,000 training iterations for u_{NN} in WANs, the DRM, and the D^{2}RM. Note that this corresponds to a total of 25,000 iterations for WANs and the D^{2}RM when taking into account the iterations dedicated to the test maximizers.

Table 4.2 displays the relative errors along different stages of the training, and Figure 4.9 shows the trial network predictions and error functions at the end of training.

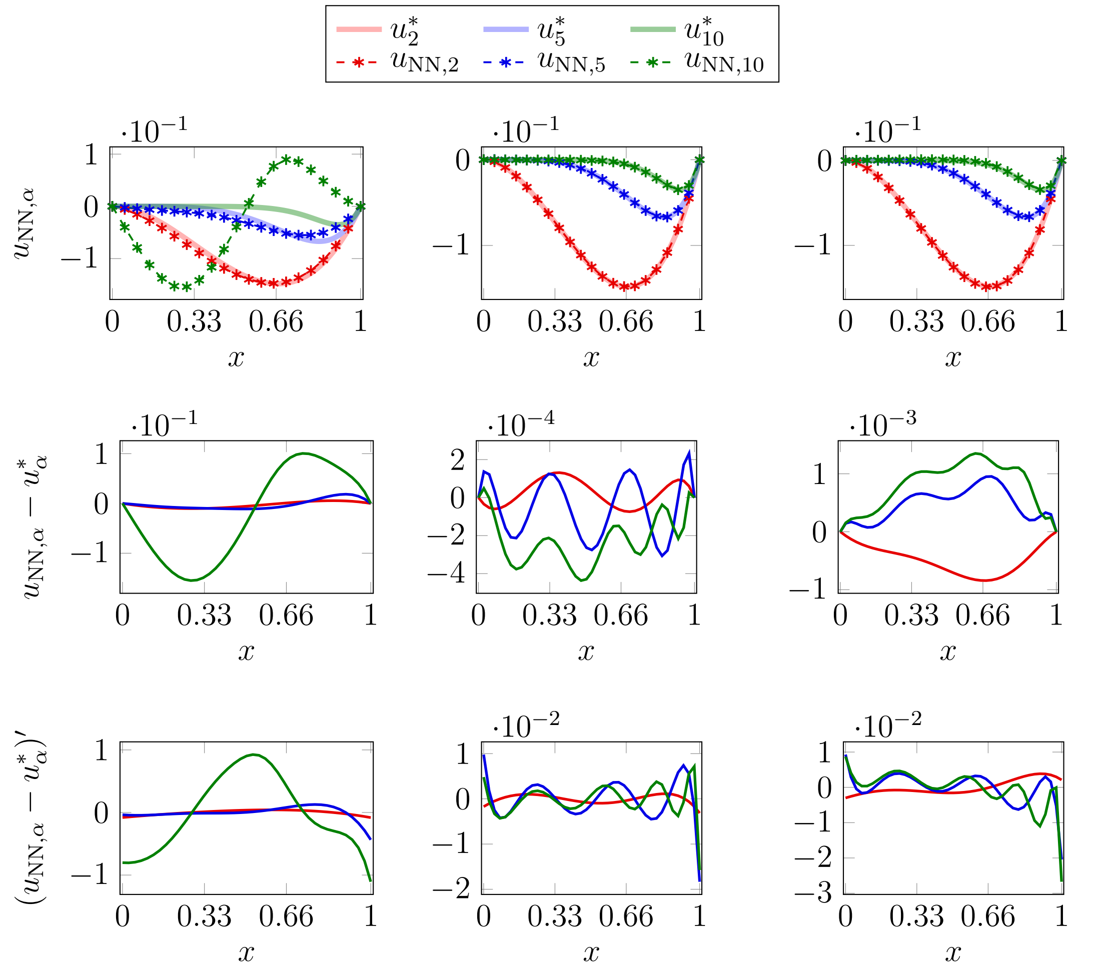
WANs show poor results in all the cases, with a clear non-convergent tendency as the training progresses, possibly justified by the unstable behavior of the method at the continuous level. Thus, we discard the WANs for the remainder of the experiments and focus on the other two methods.

In the DRM and the D^{2}RM, we observe a decreasing behavior of the relative error during training. We highlight the nearly identical behavior of the relative norm errors of u_{NN} and $\tau_{\text{NN}}(u_{\text{NN}})$ in the D^{2}RM, which suggests that the D^{2}RM behaves as the DRM, as desired.

[7]To approximate $\frac{\|u_{\text{NN}} - u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$, we perform a composite intermediate-point rule with 10^4 integration nodes for the numerator, and analytically calculate $\|u^*\|_{\mathbb{U}} = \sqrt{3}/3$ for the denominator.

Training progress		**4%**	**20%**	**40%**	**60%**	**100%**
Method	$\boldsymbol{\alpha}$	$\frac{\|u_{\mathrm{NN}}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$				
WANs	2	2.49%	3.43%	5.92%	7.40%	10.07%
	5	58.69%	41.78%	63.03%	74.06%	40.33%
	10	93.15%	93.95%	89.04%	68.85%	367.61%
DRM	2	3.40%	2.47%	1.17%	0.30%	0.23%
	5	50.50%	8.53%	2.83%	2.27%	1.59%
	10	58.55%	9.51%	3.39%	2.83%	1.69%
D^{2}RM	2	3.27%	1.84%	0.31%	0.27%	0.54%
	5	59.13%	12.93%	2.92%	2.52%	1.56%
	10	82.31%	20.18%	6.24%	2.68%	2.60%
Method	$\boldsymbol{\alpha}$	$\frac{\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}})-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$				
D^{2}RM	2	3.36%	1.94%	0.48%	0.45%	0.58%
	5	59.13%	12.93%	2.90%	2.11%	1.55%
	10	83.51%	20.18%	6.23%	2.68%	2.61%

Table 4.2: Relative errors of u_{NN} (in WANs, the DRM, and the D^{2}RM) and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ (in the D^{2}RM) along different stages of the training progress in problem (4.24) with exact solution $u^*_\alpha = x^\alpha(x-1)$ and $\alpha \in \{2, 5, 10\}$.



(a) With WANs. **(b)** With the DRM. **(c)** With the D^{2}RM.

Figure 4.9: Trial network predictions and errors for WANs, the DRM, and the D^{2}RM in model problem (4.24) with exact solution $u_\alpha^* = x^\alpha(x-1)$ for $\alpha \in \{2, 5, 10\}$.

4.5.2.2. With singularities: $1/2 < \alpha < 1$

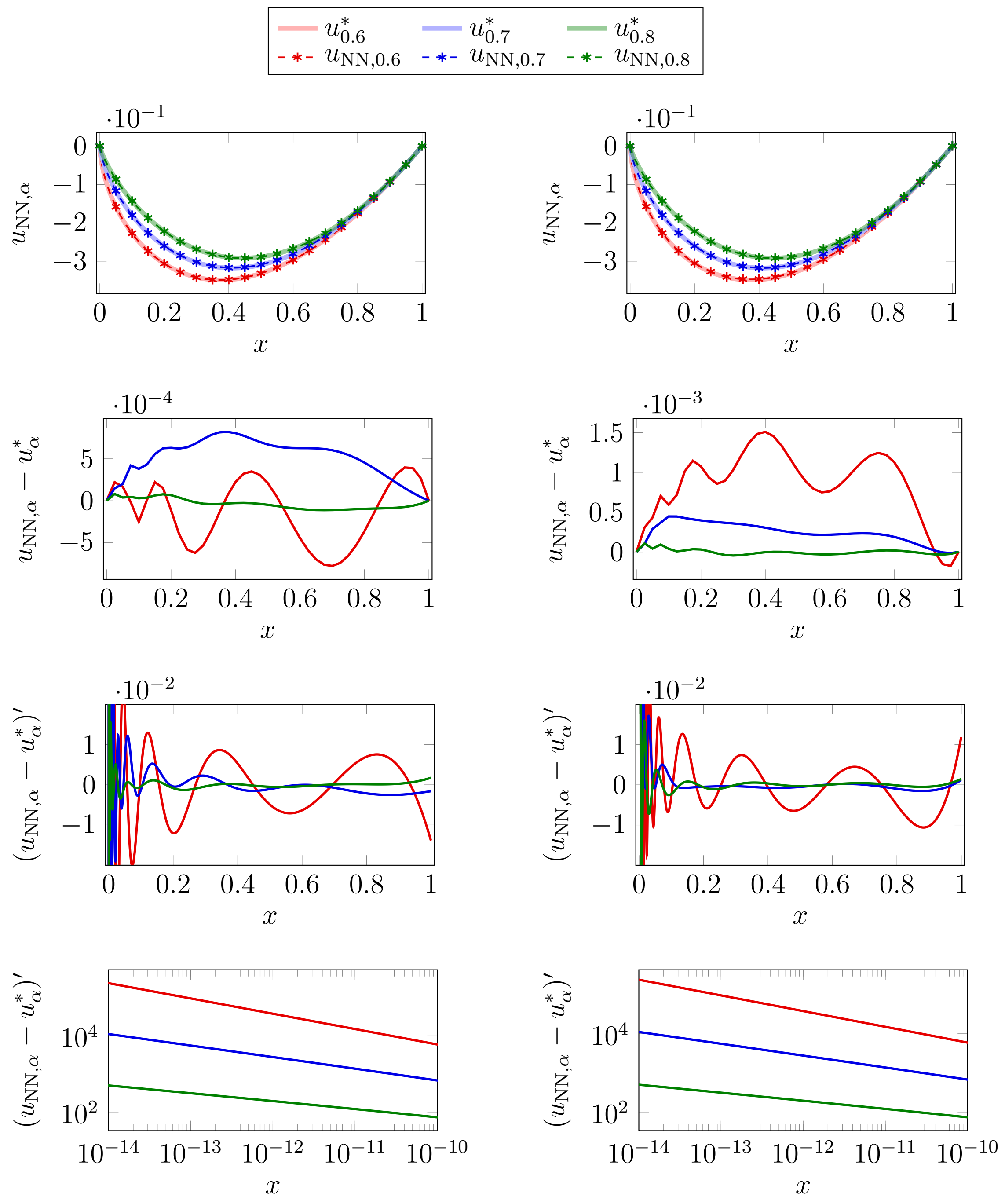
Here, $(u^*_\alpha)'(x) \to -\infty$ as $x \to 0$, which suggests that a large portion of the integration nodes should concentrate on a neighborhood of zero to appropriately capture the explosive trend of the derivative at that point. To this end, we divide the batch as the union of two equal-sized different $\beta(a, b)$ samples: one with $a = 1 = b$, and the other with $a = 10^4$ and $b = 1$. We individually experiment for $\alpha \in \{0.6, 0.7, 0.8\}$ with 100,000 iterations for approximating u_{NN}, which implies 500,000 for the D^{2}RM when taking into account the iterations dedicated to approximate the optimal test functions.

Table 4.3 displays the record of the relative error estimates along different stages of the training, and Figure 4.10 shows the u_{NN} predictions and error functions at the end of the training.

Training progress		**4%**	**20%**	**40%**	**60%**	**100%**
Method	$\boldsymbol{\alpha}$	$\frac{\Vert u_{\text{NN}} - u^*\Vert_{\mathbb{U}}}{\Vert u^*\Vert_{\mathbb{U}}} \times 100$				
DRM	0.6	42.23%	30.29%	26.91%	25.34%	23.84%
	0.7	18.47%	9.49%	7.42%	6.65%	5.95%
	0.8	8.78%	3.50%	2.46%	2.10%	1.81%
D^{2}RM	0.6	41.67%	30.25%	27.05%	25.76%	24.40%
	0.7	14.12%	9.99%	7.91%	6.98%	6.15%
	0.8	6.00%	3.62%	2.78%	2.55%	2.13%
Method	$\boldsymbol{\alpha}$	$\frac{\Vert \tau_{\text{NN}}(u_{\text{NN}}) - Tu^*\Vert_{\mathbb{V}}}{\Vert Tu^*\Vert_{\mathbb{V}}} \times 100$				
D^{2}RM	0.6	41.68%	30.25%	27.05%	25.77%	24.40%
	0.7	14.12%	10.00%	7.91%	6.98%	6.15%
	0.8	6.01%	3.62%	2.78%	2.54%	2.13%

Table 4.3: Relative errors of u_{NN} (in the DRM and the D^{2}RM) and $\tau_{\text{NN}}(u_{\text{NN}})$ (in the D^{2}RM) along different stages of the training progress in problem (4.24) with exact solution $u^*_\alpha = x^\alpha(x-1)$ and $\alpha \in \{0.6, 0.7, 0.8\}$.

The DRM and the D^{2}RM decrease the relative errors at similar rates, slowing down notably from the 40% of the training progress onwards. For $\alpha \in \{0.7, 0.8\}$, we achieve final relative errors below 7.5% at the end of the training, with loss predictions around -0.3632 and -0.2562 where the optimal values are around -0.3646 and -0.2564, respectively. For $\alpha = 0.6$, the relative error is above 20%, with a final loss prediction around -0.6429 where the optimal value is around -0.6818. Intending to concentrate more points around the singularity, we



(a) With the DRM

(b) With the D^{2}RM

Figure 4.10: Trial network predictions and errors for the DRM and the D^{2}RM in model problem (4.24) with exact solution $u^*_\alpha = x^\alpha(x-1)$ for $\alpha \in \{0.6, 0.7, 0.8\}$. The last row is an augmented version of the third one in a reduced neighborhood of zero.

performed several experiments modifying the a parameter of the beta distribution, but without success.

4.5.3. Pure diffusion with Dirac delta source

We now select the source $l = 4\delta_{1/2} \in H^{-1}(0,1) \setminus (L^2(0,1))'$, where $\delta_{1/2}$ denotes the Dirac delta distribution at 1/2, i.e., $\delta_{1/2}(v) = v(1/2)$ for all $v \in H_0^1(0,1)$. The corresponding exact solution is

$$u^* = \begin{cases} 2x, & \text{if } x < 1/2, \\ 2(1-x), & \text{if } x > 1/2. \end{cases} \tag{4.29}$$

We train u_{NN} for 20,000, selecting $a = 1 = b$ and $a = 10 = b$ in the $\beta(a,b)$ probability distribution for the two-sampled batch. Table 4.4 records the relative errors at three different stages of the training progress. Figure 4.11 shows the trial network predictions, errors, and derivative of the errors at those checkpoints. As we observe, the D^{2}RM shows higher errors than the DRM.

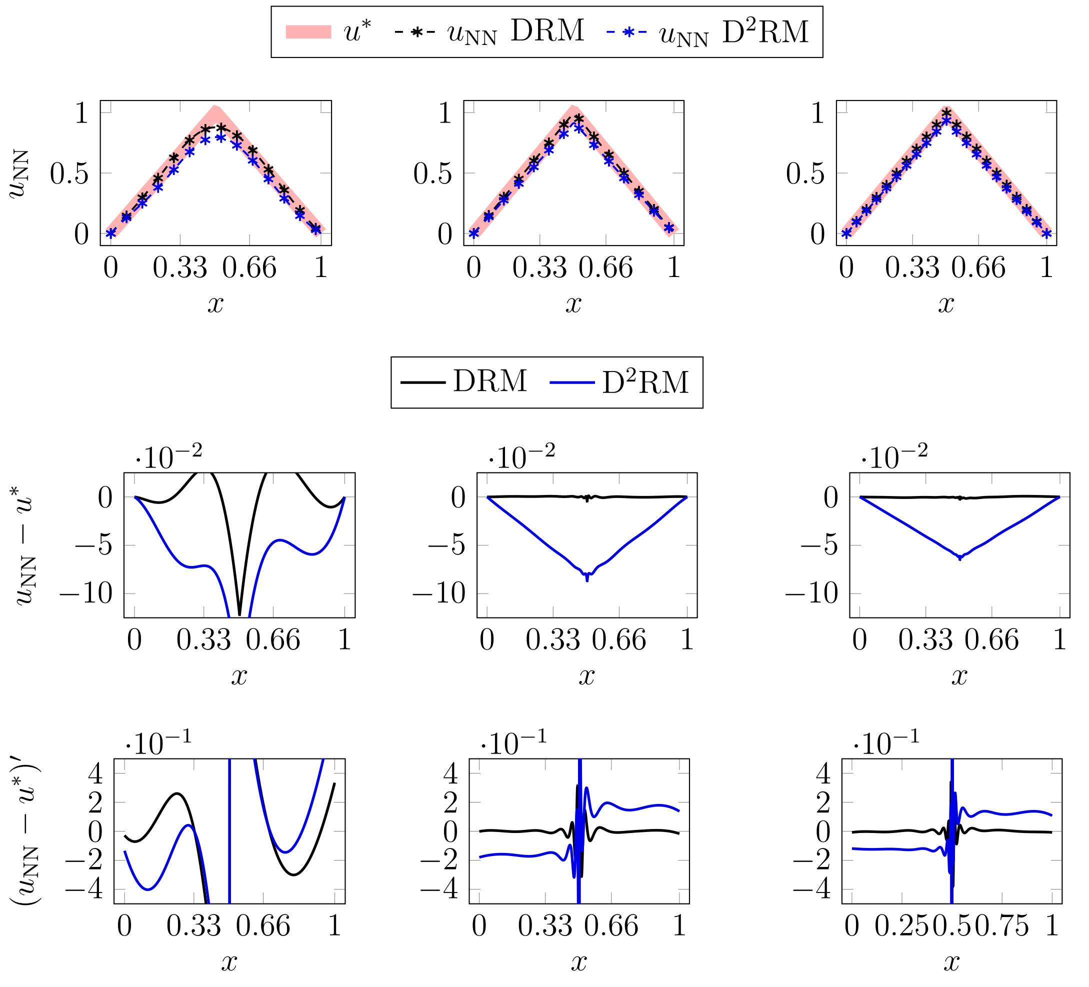
Training progress	**4%**	**20%**	**40%**	**60%**	**100%**
Method	$\frac{\|u_{\text{NN}}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$				
DRM	32.32%	10.10%	6.22%	5.08%	4.34%
D^{2}RM	32.43%	13.95%	10.87%	9.41%	7.95%
Method	$\frac{\|\tau_{\text{NN}}(u_{\text{NN}})-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$				
D^{2}RM	32.68%	14.09%	10.84%	9.53%	8.16%

Table 4.4: Relative errors of u_{NN} (in the DRM and the D^{2}RM) and $\tau_{\text{NN}}(u_{\text{NN}})$ (in the D^{2}RM) along different stages of the training progress in model problem (4.24) with exact solution (4.29).

4.5.4. Pure convection with Dirac delta source

We consider the spatial convection equation

$$\begin{cases} u' = \delta_{1/2}, & \text{in } (0,1), \\ u(0) = 0. \end{cases} \tag{4.30}$$



(a) 4% of the training. **(b)** 40% of the training. **(c)** 100% of the training.

Figure 4.11: Trial error functions in model problem (4.24) with with solution (4.29) at different stages of the training progress for the DRM and the D^{2}RM.

Here, the (ultra)weak formulation is appropriate because $\delta_{1/2}$ does not belong to $(L^2(0,1))'$. Integration by parts yields:

$$b(u,v) := -\int_0^1 uv', \qquad l(v) := v(1/2), \qquad u \in \mathbb{U}, v \in \mathbb{V}, \tag{4.31}$$

with $\mathbb{U} = L^2(0,1)$ and $\mathbb{V} = H^1_{0)} := \{v \in H^1(0,1) : v(1) = 0\}$. The trial-to-test operator is no longer the identity and has the following integral form [80, 159]:

$$(Tu)(x) = \int_x^1 u(s)ds, \qquad u \in L^2(0,1). \tag{4.32}$$

For the exact solution and its corresponding optimal test function, we have

$$u^* = \begin{cases} 0, & \text{if } 0 < x < 1/2, \\ 1, & \text{if } 1/2 < x < 1, \end{cases} \in L^2(0,1), \tag{4.33a}$$

$$Tu^* = \begin{cases} 1/2 & \text{if } 0 < x < 1/2, \\ 1-x, & \text{if } 1/2 < x < 1, \end{cases} \in H^1_{0)}(0,1). \tag{4.33b}$$

In our context of NNs, considering T as an available operator is challenging, so we discard employing the Generalized Deep Ritz Method (GDRM) here, and alternatively employ the (DRM)$'$—recall (4.23)—and the D^{2}RM:

- **Adjoint Deep Ritz Method.** Following item (c) in Section 4.2.4, we minimize $\mathcal{F}'$ to find an approximation to the optimal test function of the trial solution, i.e.,

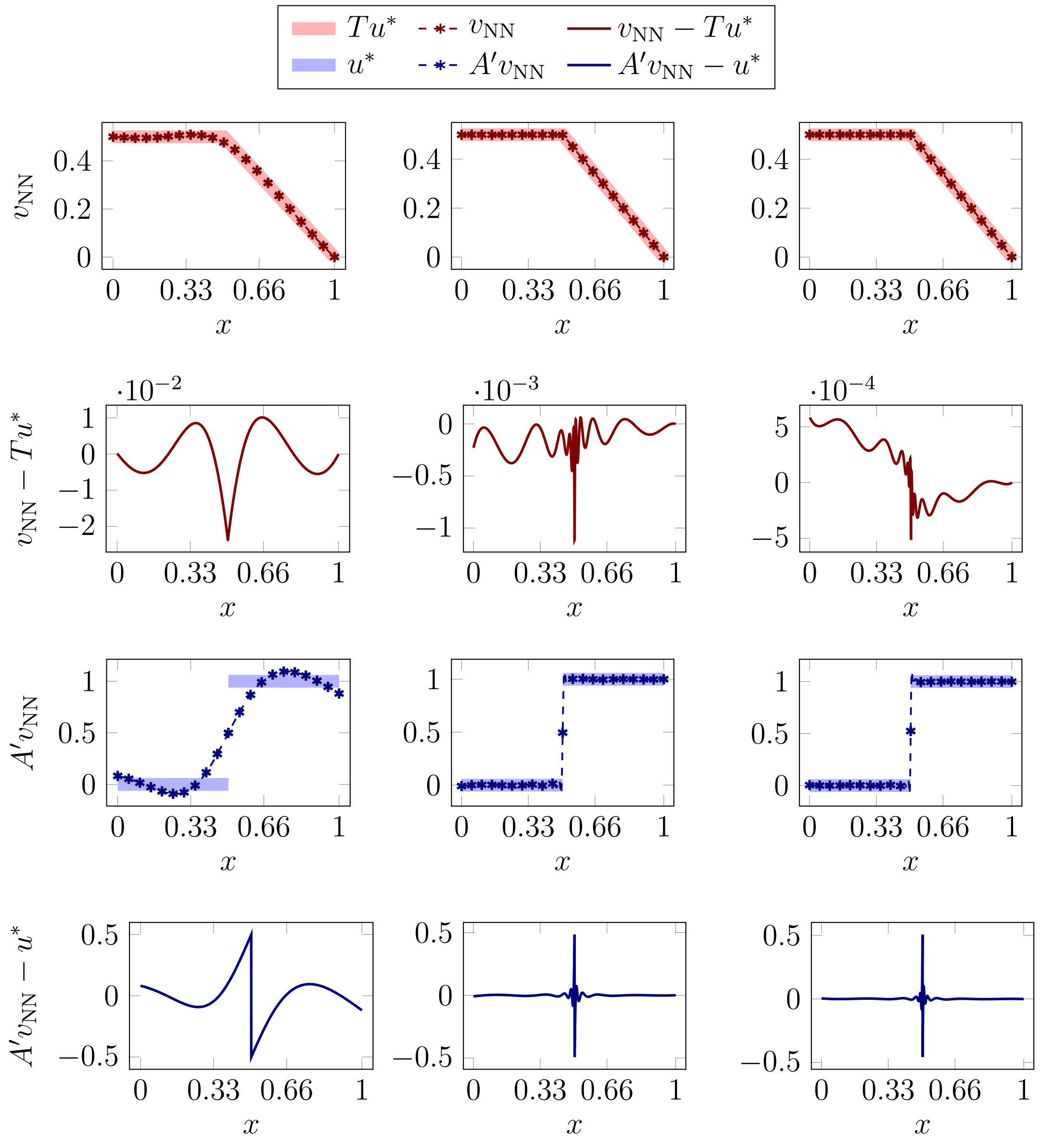
  $$Tu^* \approx \arg \min_{v_{\mathrm{NN}} \in \mathbb{V}_{\mathrm{NN}}} \mathcal{F}'(v_{\mathrm{NN}})^{8}. \tag{4.34}$$

  Then, we post-process the obtained minimizer by applying the available adjoint operator $A' = -d/dx$.

  We perform 50,000 training iterations. Table 4.5 records the evolution of the approximated relative norm errors $\frac{\|v_{\mathrm{NN}} - Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}}$ and $\frac{\|A'v_{\mathrm{NN}} - u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}}$ along the training progress. Figure 4.12 shows the predictions and errors of both v_{NN} and $A'v_{\mathrm{NN}}$ along different stages of the training.

- **Deep Double Ritz Method.** We impose the outflow-boundary condition only to τ_{NN}, letting u_{NN} be boundary-free, and perform 50,000 iterations for u_{NN} and increase the number of inner-loop iterations from four to nine (i.e., a total of 500,000 iterations if we account those dedicated to the test functions). Table 4.6 records the relative errors for u_{NN} and $\tau_{\mathrm{NN}} u_{\mathrm{NN}}$ along different stages of the training. Figure 4.13 shows the trial and optimal test network predictions and error functions at the end of the training.

[8]The "arg" and "min" terms should be understood loosely according to the possible lack of existence discussed in Chapter 2.



(a) 4% of the training. **(b)** 40% of the training. **(c)** 100% of the training.

Figure 4.12: Test network predicitions v_{NN}, post-processings $A'v_{\mathrm{NN}}$, and error functions for the (DRM)′ in model problem (4.30) with exact solution (4.33) along different stages of the training progress.

Training progress	**4%**	**20%**	**40%**	**60%**	**100%**
$\frac{\lVert v_{\text{NN}} - Tu^* \rVert_{\mathbb{V}}}{\lVert Tu^* \rVert_{\mathbb{V}}} \times 100$	19.03%	4.58%	3.55%	3.20%	2.83%
$\frac{\lVert A' v_{\text{NN}} - u^* \rVert_{\mathbb{U}}}{\lVert u^* \rVert_{\mathbb{U}}} \times 100$	21.95%	5.28%	4.10%	3.69%	3.27%

Table 4.5: Relative errors of v_{NN} and $A'v_{\text{NN}}$ along different stages of the training progress in problem (4.30) with exact solution (4.33).

Training progress	**4%**	**20%**	**40%**	**60%**	**100%**
$\frac{\lVert u_{\text{NN}} - u^* \rVert_{\mathbb{U}}}{\lVert u^* \rVert_{\mathbb{U}}}$	42.05%	29.69%	18.42%	11.20%	8.93%
$\frac{\lVert \tau_{\text{NN}}(u_{\text{NN}}) - Tu^* \rVert_{\mathbb{V}}}{\lVert Tu^* \rVert_{\mathbb{V}}}$	37.27%	28.25%	19.02%	9.55%	6.13%

Table 4.6: Relative errors of u_{NN} and $\tau_{\text{NN}}(u_{\text{NN}})$ along different stages of the training progress in problem (4.30) with exact solution (4.33).

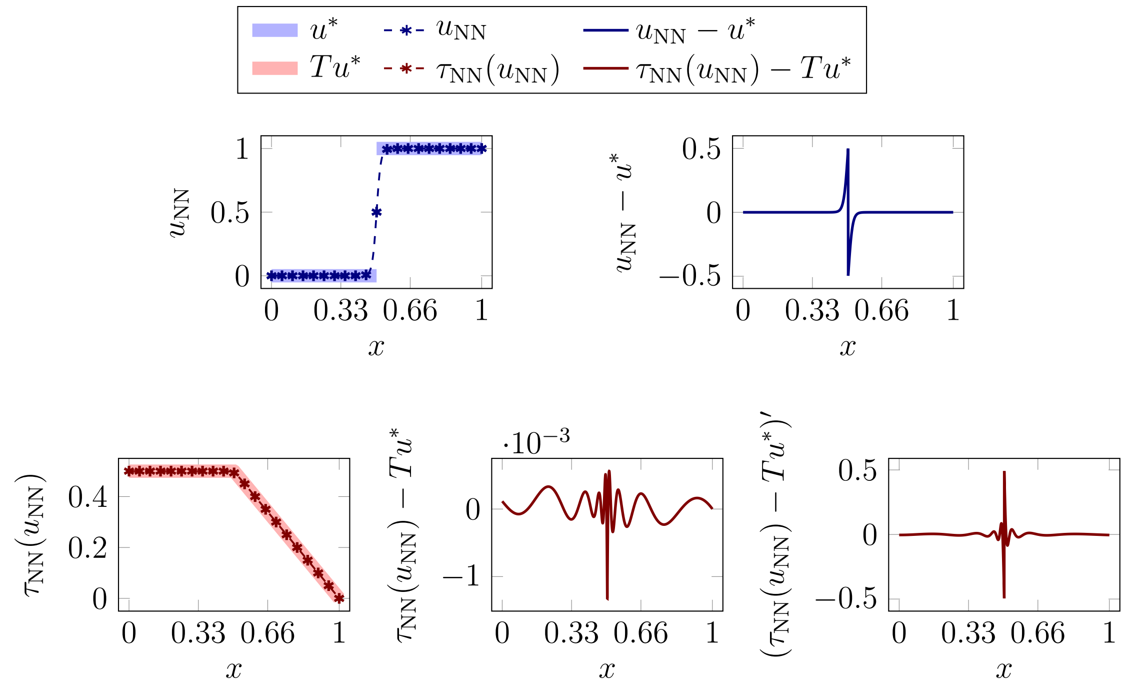


Figure 4.13: u_{NN} and $\tau_{\text{NN}}(u_{\text{NN}})$ predictions, errors, and derivative of the errors for the D^{2}RM in problem (4.30) with exact solution (4.33).

4.5.5. Pure convection in 2D

Let

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial u}{\partial y} = k\pi \sin\left(k\pi(x+y)\right), & \text{in } \Omega = (0,1)\times(0,1), \\ u(x,0) = u(0,y) = 0, & 0 \leq x, y \leq 1, \end{cases} \tag{4.35}$$

and consider its strong variational formulation

$$b(u,v) := \int_\Omega \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) v, \quad l(v) := k\pi \int_\Omega \sin(k\pi(x+y))v, \tag{4.36}$$

for $u \in \mathbb{U}$ and $v \in \mathbb{V}$ such that

$$\mathbb{U} = \left\{ u \in L^2(\Omega) : \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \in L^2(\Omega) \text{ and } u(x,y) = 0 \text{ when } xy = 0 \right\}, \tag{4.37a}$$

$$\mathbb{V} = L^2(\Omega). \tag{4.37b}$$

Its exact solution is $u^* = \sin(kx)\sin(ky)$ and the trial-to-test operator is the PDE operator.

For $k = 3/2$, we perform 200,000 iterations in the D^{2}RM with a training regime of nine iterations in the inner loop for each iteration in the outer loop. Exceptionally, we run the inner loop for 2,000 iterations before the first outer-loop iteration. For integration, we consider 50 nodes on each axis (i.e., 250 integration points on the entire domain due to the cartesian-product structure). We increase the NN architecture to three layers of 50-neuron width.

Figure 4.14 shows the trial and optimal test predictions with corresponding error functions at the end of the training. The resulting relative errors are 3.62% and 1.70% for u_{NN} and $\tau_{\text{NN}}(u_{\text{NN}})$, respectively.

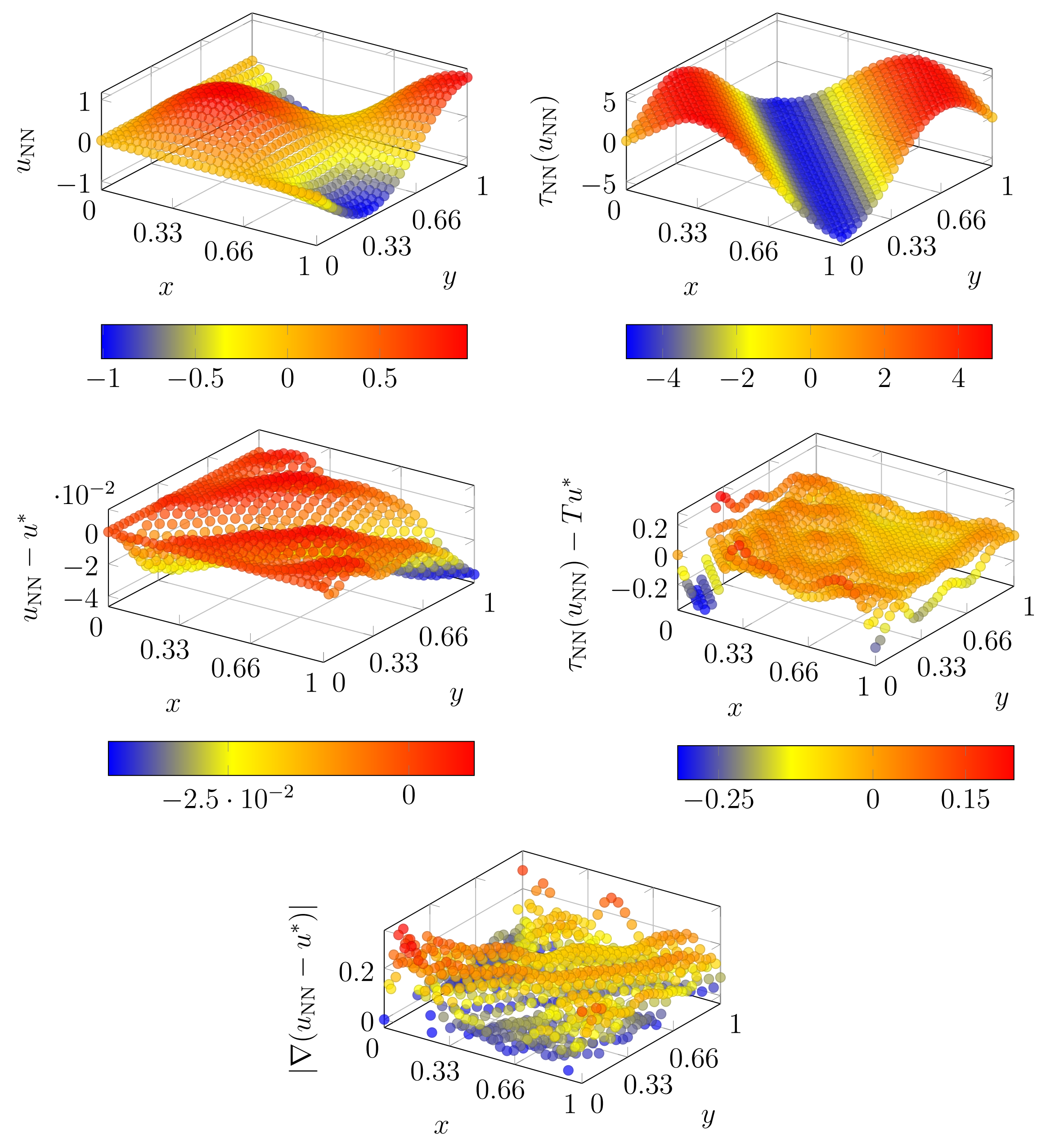


Figure 4.14: Trial and optimal test predictions, errors, and derivative of the trial error for the D^{2}RM in model problem (4.30) with exact solution (4.33).

5. Memorian onarritutako Monte Carlo integrazioa[*]

Laburpena. Monte Carlo integrazioa oso erabilia den koadratura-erregela bat da neurona-sareen bidez deribatu partzialetako ekuazioak ebazten direnean, gain-doikuntza gabeko hurbilpenak bermatzen dituelako eta dimentsio handiko eskalagarritasun gaitasuna duelako. Hala ere, metodo estokastiko horrek galerak eta gradiente zaratatsuak eragiten ditu entrenamenduan zehar, konbergentzia-diagnostiko egokia zailago eginez. Normalean, hori gainditzeko integrazio-puntu kopuru izugarria erabiltzen da, nahiz eta horrek entrenamenduaren errendimendua hondatu. Lan honetan, memoria oinarri duen Monte Carlo integrazio-metodo bat proposatzen da, koadratura-erregela zehatza sortzea ahalbidetzen duena eta entrenamendurako ez duena behar lagin handien prozesakuntzak eragiten duen kostu konputazional garestirik. *Argitaratutako bertsiorako, ikus [218].*

5.1. Sarrera

NNaren portaera txarra ekar dezake integrazio-puntuetatik urrun integrazio-puntu finkoak dituen koadratura-erregela deterministiko bat erabiltzeak, gaindoikuntzako fenomenoagatik [186]. Horrek, integrazio-errore esanguratsuak eragiten ditu eta, ondorioz, pobreak dira lortzen diren hurbilpenak. Hori gainditzeko, egokia da *Monte Carlo (MC) integrazioa* koadratura-erregelaren aukeraketa, laginketa estokastikoa eta sarerik gabeko egitura baititu [128, 84, 62, 48]. Hala ere, Monte Carlo integrazio-errorea honako ordena honetakoa da: $\mathcal{O}(1/\sqrt{N})$, non N integrazio-puntuen kopurua baita [163]. Orduan, praktikan, hamarnaka mila edo ehunka mila integrazio-puntu behar dira MC integrazioaren bidez errore-maila onargarri bat lortzeko (baita dimentsio bakarreko problemetarako ere), entrenamendu-abiaduraren kaltetan.

Lan honetan, memorian oinarritutako koadratura-erregela bat proposatuko dugu, NNak inplikatzen dituen integral definituak hurbiltzen dituena eta entrenamenduan zehar aurreko iterazioetan lortutako informazioa erabiliko duena.

[*]See Appendix C for the English version/Ikus C. eranskina ingelesezko bertsiorako.

Integral horietatik espero den balioa nabarmen aldatzen ez den bitartean, teknika honek espero den integrazio-errorea murriztu egiten du eta koadratura-erregela hobeak eragiten ditu. Gainera, gradientea ere integral definitu baten bidez adieraz daitekeenez, ikuspegi hori gradientearen konputazioei ere aplikatuko diegu, optimizatzailearen hiperparametroak behar bezala doituz. Horrela, *momentum* metodo ezagunaren berrinterpretazio bat lortuko dugu [177].

Kapitulu honen gainerako partea honela dago antolatuta. 5.2. atalean hurbilketa- eta optimizazio-esparruak berrikusiko dira, 5.3. atalean memorian oinarritutako integrazio- eta optimizazio-estrategiak proposatuko dira, eta 5.4. atalean proposamen horiek *momentum* metodoarekin lotuko dira.

5.2. Hurbiltzeko, diskretizatzeko, parametrizatzeko eta optimizatzeko konfigurazioen berrikuspena

2. kapituluan landutako kontzeptu asko laburbilduko ditugu notazioa (bir)definitzeko, eta irakurlearentzat modu arinagoan eta errazagoan aurkeztuko dugu metodoa.

Izan bedi

$$u^* = \arg\min_{u \in \mathbb{U}} \mathcal{F}(u), \tag{5.1}$$

ondo definituta dagoen minimizazio problema bat, non $\mathbb{U}$-k funtzioen bilaketa-espazioa adierazten baitu, Ω domeinua dutenak. $\mathcal{F} : \mathbb{U} \longrightarrow \mathbb{R}$ minimizazio problema zuzentzen duen funtzio objektiboa da eta u^* haren soluzio zehatz bakarra da.

Izan bedi $u_\theta : \Omega \longrightarrow \mathbb{R}$ neurona-sareko arkitektura bat, hurbilpenaren parametrizazioa adierazten duena, non $\theta \in \Theta$ entrenagarriak diren parametroen multzoa baita (Θ-k parametroen domeinua adierazten du). Orduan, (5.1) problemaren hurbilpena egiteko u_θ-ren bidez, $\mathbb{U}$ bilaketa-espazioren ordez $\mathbb{U}_\Theta = \{u_\theta : \theta \in \Theta\}$ espazio parametrizatura mugatuko gara.

Minimizazioa gauzatzeko, lehen mailako gradiente-jaitsiera eskema bat erabiliko dugu. Hona hemen eskema hori:

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t}), \tag{5.2}$$

non $\lambda > 0$ ikaskuntza-tasa baita eta θ_t-k entrenagarriak diren parametroen multzoa adierazten baitu t. iterazioan.

Funtzio objetiboak, $\mathcal{F}$-k, integral definituaren itxura badu, hau da,

$$\mathcal{F}(v_\theta) = \int_\Omega I(v_\theta)(x)\; dx, \tag{5.3}$$

orduan koadratura-erregela baten bidez hurbilduko dugu, $\mathcal{L}$ *galera-funtzioa* sortuz. MC koadratura-erregela hautatzen badugu, galera-funtzioak honako itxura hau du:

$$\mathcal{F}(v_\theta) \approx \mathcal{L}(v_\theta) := \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} I(v_\theta)(x_i), \tag{5.4}$$

non $\{x_i\}_{i=1}^N$-k integrazio-puntuen multzoa adierazten baitu, Ω-ko zorizko banaketa uniforme batetik lagindutakoa.

Era berean, gradienteei dagokienez, honako itxura hau dugu:

$$\frac{\partial \mathcal{F}}{\partial \theta}(v_\theta) \approx g(v_\theta) := \frac{\partial \mathcal{L}}{\partial \theta}(v_\theta) = \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} \frac{\partial I(v_\theta)}{\partial \theta}(x_i). \tag{5.5}$$

Honela, (5.5)-ek (5.2)-ren bertsio diskretizatu bat adierazten du, *gradiente jaitsiera estokastikoa (SGDa)*[2] deitua [187], eta honela deskribatuta dagoena:

$$\theta_{t+1} := \theta_t - \lambda g(v_{\theta_t}). \tag{5.6}$$

Hemendik aurrera, $\mathcal{F}(\theta_t)$, $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$, $\mathcal{L}(\theta_t)$ eta $g(\theta_t)$ idatziko dugu, sinpletasunagatik, $\mathcal{F}(v_{\theta_t})$-ren, $\frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t})$-ren, $\mathcal{L}(v_{\theta_t})$-ren eta $g(v_{\theta_t})$-ren ordez, hurrenez hurren[3].

5.3. Memorian oinarritutako integrazioa eta optimizazioa

Sarea (5.6)-ren arabera entrenatzen badugu, portaera zaratatsua eta oszilakorra lortzen dugu galeran eta gradientean. Hori gertatzen da Monte Carlo integrazioak entrenamenduko iterazio bakoitzean txertatzen duen erroreagatik. 5.1. irudiak Monte Carlo integrazioaren portaera zaratatsua erakusten du (kurba urdina), $\mathcal{F}$ zehazki kalkulatzea ahalbidetzen duen parametro entrenagarri bakarreko sare batean (kurba beltza).

Atal hau hiru zatitan antolatuta dago. 5.3.1. atalean esperimentaziorako aukeratutako eredu problema aurkezten da, 5.3.2. atalean proposatutako memorian oinarritutako MC integrazio-araua deskribatzen da, eta 5.3.3. atalean memorian oinarritutako eskema hori gradienteetara hedatzen da entrenamendurako.

[2]Ingelesetik, *Stochastic Gradient Descent (SGD)*.

[3]Ohartu errealizazio-maparen terminologia alde batera utzi dugula —gogoratu (2.5)— parametroen funtzio objektiboaren mendekotasuna adieraztean. Hau da, $\mathcal{F}(\theta)$-k $(\mathcal{F} \circ \Phi_{\mathrm{NN}})(\theta)$ ordezkatzen du, 2. kapituluan garatutako notazioaren arabera.

5.3.1. Eredu problema

Izan bedi $-u'' = 4\delta_{1/2}$, $\Omega = (0,1)$ domeinuan eta $\partial\Omega = \{0,1\}$-en Dirichlet muturbalio homogeneoekin. Orduan, bere formulazio bariazional ahulak honako forma bilineal eta lineal hauek ditu:

$$b(u,v) = \int_0^1 u'v', \qquad l(v) = v(1/2), \qquad u \in \mathbb{U} = H_0^1(0,1) = \mathbb{V} \ni v, \tag{5.7}$$

non bere soluzio zehatza honako hau baita: $u^*(x) = 2x$, $0 \leq x \leq 1/2$ bada; eta $u^*(x) = 2(1-x)$, $1/2 \leq x \leq 1$ bada. Gainera, Ritz motako minimizazio-birformulazio bat onartzen du, honela:

$$u^* = \arg\min_{u\in\mathbb{U}} \frac{1}{2}\int_0^1 [u'(x)]^2 dx - u(1/2). \tag{5.8}$$

Orain, $2\tanh(\theta(x-1/2))$-k $(u^*)'$ arbitrarioki hurbiltzeko gaitasuna duela ohartuz —gogoratu 2.4. atala—, analitikoki integratuz eta mutur-balioak ezarriz, hurrengo NN arkitektura (atipikoa) lortzen dugu u_θ-k hurbilketa arbitrarioa izan dezan u^*-rekiko:

$$xu_\theta(x) = \frac{2}{\theta}\Big\{\log\big[\cosh(\theta/2)\big] - \log\big[\cosh(\theta(1/2-x))\big]\Big\}. \tag{5.9}$$

Ondorioz, kalkulu errazen bidez honako hauek lortzen ditugu:

$$\mathcal{F}(\theta) = 2 - \frac{4}{\theta}\tanh(\theta/2) - \frac{8}{\theta}\log\big[\cosh(\theta/2)\big], \tag{5.10a}$$

$$\frac{\partial\mathcal{F}}{\partial\theta}(\theta) = \frac{-4(\theta-1)\tanh(\theta/2) - 2\theta\operatorname{sech}^2(\theta/2) + 8\log[\cosh(\theta/2)]}{\theta^2}, \tag{5.10b}$$

$\theta \neq 0$ denean. Konturatu konbergentzia/hurbiltasuna lortzen dela $\theta \to \infty$ denean, hurrengo ondoriozkoekin: $\mathcal{F}(\theta) \to -2$ eta $\frac{\partial\mathcal{F}}{\partial\theta}(\theta) \to 0$.

5.3.2. Integrazioa

Entrenamenduan integrazio-errorea murrizteko, (5.4)-ren ordez, honako errepikapen-prozesu hau erabiliko dugu:

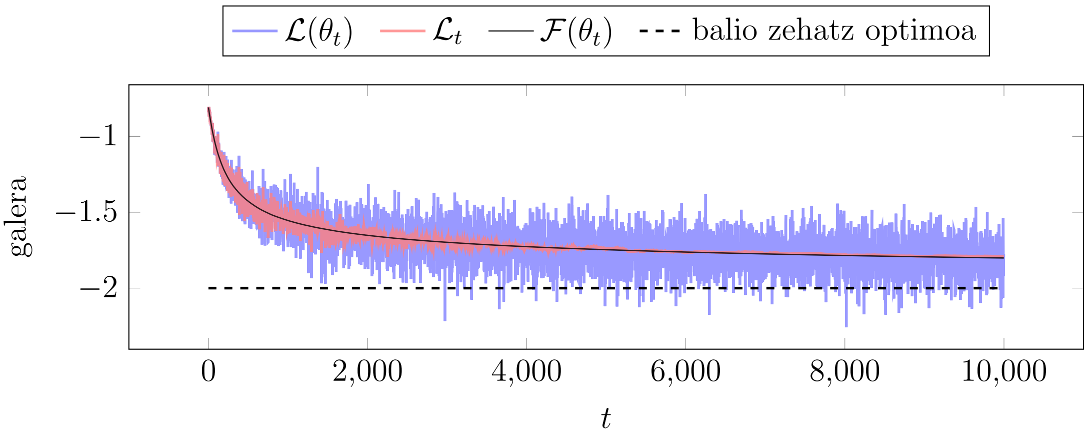
$$\mathcal{F}(\theta_t) \approx \mathcal{L}_t := \begin{cases} \mathcal{L}(\theta_0), & t = 0 \text{ bada}, \\ \alpha_t\mathcal{L}(\theta_t) + (1-\alpha_t)\mathcal{L}_{t-1}, & t \geq 1 \text{ bada}, \end{cases} \tag{5.11a}$$

non $\mathcal{L}(\theta_t)$ MCren integrazio zenbatespena baita t. iteraziorako, eta $\{\alpha_t\}_{t\geq 0}$ aukeratutako koefizienteen segida bat izanik $0 < \alpha_t \leq 1$ eta $\alpha_0 = 1$ betetzen dituena. Modu hedatuan, honako itxura hau du:

$$\mathcal{L}_t = \sum_{l=0}^{t} \alpha_l \left(\prod_{s=1}^{t-l}(1-\alpha_{l+s})\right)\mathcal{L}(\theta_l). \tag{5.11b}$$

Horrek erakusten du $\mathcal{L}_t$ $\mathcal{F}(\theta_t)$-ren zenbatespena oraingo eta aurreko MC integrazio zenbatespen guztien konbinazio lineala dela. Ohartu, $t \geq 0$ guztietarako $\alpha_t = 1$ bada, orduan memoriarik gabeko ohiko MC integrazio kasua berreskuratzen dugula.

5.1. irudiaren kurba gorriak erakusten du memorian oinarritutako $\mathcal{L}_t$ galera-funtzioaren eboluzioa entrenamenduan zehar, SGD optimizazio arruntaren, (5.6)-ren, arabera egiten denean eta hurrengo koefizienteen hautaketarekin: $\alpha_t = e^{-0.001t} + 0.001$. Hasieran, errore handiekin integratzen dugu (α_t ia bat da, eta, beraz, ia ez dago memoriarik $\mathcal{L}_t$-n). Hala ere, entrenamenduan aurrera egin ahala, memoria gehitzen diogu $\mathcal{L}_t$-ri (α_t txikituz), eta, ondorioz, integrazio-errorea murriztu egiten da. $\mathcal{L}_t$-k hurbilketa zehatzagoak egiten ditu $\mathcal{L}(\theta_t)$-k baino, konbergentziaren jarraipen hobea egiteko aukera eskainiz, adibidez, entrenamenduan zehar gelditzeko irizpide egokiak ezartzekoa.



5.1. irudia: 5.3.1. atalean aukeztutako NNaren entrenamendua, parametro entrenagarri bakarrekoari dagokiona, eta haren arkitekturak $\mathcal{F}$ zehazki kalkulatzeko aukera ematen duena. Entrenamendua (5.6)-ren arabera egiten da, eta, beraz, $\mathcal{L}(\theta_t)$ da entrenamendurako galera. $\mathcal{L}_t$ eta $\mathcal{F}(\theta_t)$ bakarrik jarraipena egiteko kalkulatzen dira.

5.3.3. Optimizazioa

Nahiz eta proposatutako eskemak $\mathcal{F}$-ren hurbilketa hobetzen duen, erabiltzen den SGD eskema (5.11)-n SGD klasikoaren baliokidea da ikaskuntza-tasa α_t-rekin biderkatzen denean.

Hala ere, memorian oinarritutako integrazioaren eskema gradienteei eman diezaiekegu, $g(\theta_t)$ MC integrazioaren bidez lortzen delako —gogoratu (5.5)—. Beraz, gradienteentzako memoria-eskema honela geratuko litzateke:

$$\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t) \approx g_t := \begin{cases} g(\theta_0), & t = 0 \text{ bada}, \\ \gamma_t g(\theta_t) + (1-\gamma_t) g_{t-1}, & t \geq 1 \text{ bada}, \end{cases} \tag{5.12a}$$

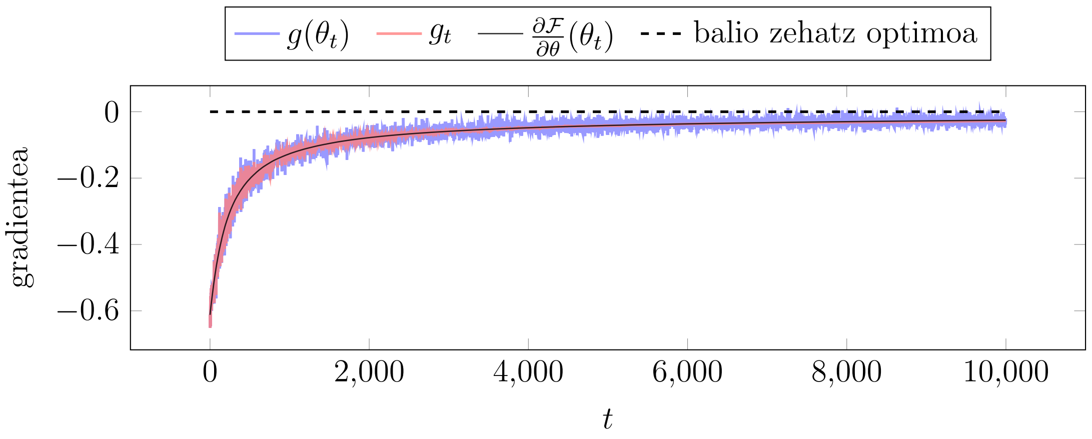
non $\{\gamma_t\}_{t\geq 0}$ aukeratutako koefizienteen segida baita $0 < \gamma_t \leq 1$ eta $\gamma_0 = 1$ izanik. Orduan, memorian oinarritutako SGD optimizatzaile bat lortzen dugu, $g(\theta_t)$ terminoaren ordez g_t terminoa erabiltzen duena —gogoratu (5.6)—; hau da,

$$\theta_{t+1} := \theta_t - \lambda g_t. \tag{5.12b}$$

Modu hedatuan, honako itxura hau du:

$$g_t = \sum_{l=0}^{t} \gamma_l \left(\prod_{s=1}^{t-l} (1 - \gamma_{l+s}) \right) g(\theta_l). \tag{5.12c}$$

5.2. irudiak gradientearen eboluzioa erakusten du 5.1. irudiko baldintzetan. Lehen bezala, gainbehera esponentziala erabili dugu, $\alpha_t = \gamma_t$ aukeratuz $t \geq 0$ denean. g_t-k gradiente zehatzen hurbilketa zehatzagoak ematen ditu $g(\theta_t)$-k baino, portaera zaratatsua minimizatuz.



5.2. irudia: Gradientearen eboluzioa 5.1. irudiko entrenamenduaren baldintzetan. Optimizazioa (5.6)-ren arabera egin da $g(\theta_t)$ erabiliz. g_t eta $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$ kalkulatu dira jarraipenerako.

α_t eta γ_t koefizienteen doikuntza egokia funtsezkoa da integrazioaren errendimendua maximizatzeko. Koefizienteak altuak (memoria baxua) izan behar dira inplikatutako integralak azkar aldatzen direnean (adibidez, entrenamenduaren hasieran). Aitzitik, hurbilpena orekatik gertu dagoenean eta inplikatutako integralak poliki-poliki aldatzen direnean, koefizienteak baxuak (memoria altua) izan behar dira.

5.4. Momentum metodoarekiko lotura

Momentum duen SGD optimizatzailea (SGDM)[4] [177, 209] hurrengo bi urratseko metodo errekurtsiboaren arabera aurkezten da:

$$v_{t+1} := \beta v_t - g(\theta_t), \tag{5.13a}$$

$$\theta_{t+1} := \theta_t + \lambda v_{t+1}, \tag{5.13b}$$

non v_t *momentum metagailu-tasa* baita $v_0 = 0$ balioarekin hasiaraziz, eta $0 \leq \beta < 1$ *momentum-koefizientea* izanik. $\beta = 0$ bada, SGD optimizatzaile arrunta berreskuratzen dugu (5.6). (5.13) erregela berridazten badugu $\theta_{t+1} = \theta_t - \lambda g_t$ eskemaren arabera, honako hau daukagu:

$$g_t = g(\theta_t) + \beta g_{t-1} = \sum_{l=0}^{t} \beta^{t-l} g(\theta_l). \tag{5.13c}$$

SGDMaren bertsio sofistikatuago bat entrenamenduan zehar momentum-koefizientea aldatzean datza (ikus, adibidez, [50]); hau da, (5.13) araberako erregela jarraituz baina β-ren ordez $\beta_t \in [0,1)$ erabilita, $\{\beta_t\}_{t\geq 1}$ hautatutako segidarako. Orduan, g_t terminoak honako itxura hau du:

$$g_t = g(\theta_t) + \beta_t g_{t-1} = \sum_{l=0}^{t} \left(\prod_{s=1}^{t-l} \beta_{l+s} \right) g(\theta_l). \tag{5.14}$$

Entrenamenduan zehar β_t hiperparametro egokiak hautatzea erronka bat da. Hala ere, SGDM optimizatzailearen ikaskuntza-tasa eta momentum-koefizientea berregokituz, γ_t-ren arabera $t \geq 1$ denean —gogoratu (5.12)—, honela:

$$\lambda_t := \lambda_{t-1} \frac{\gamma_t}{\gamma_{t-1}}, \qquad \lambda_0 := \lambda, \tag{5.15a}$$

$$\beta_t := \gamma_{t-1} \frac{1-\gamma_t}{\gamma_t}, \tag{5.15b}$$

orduan memorian oinarritutako proposamena berreskuratzen dugu (5.12).

Bi optimizazioek, (5.12)-k eta (5.13)–(5.14)-k, estokastikoki gradienteak metatzen dituzte parametro entrenagarriak berregokitzeko. Hala ere, (5.13)–(5.14)-k aurreko gradienteen batezbesteko geometrikoa hartzen du kontuan, g_t-k $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$-ren antza hartzeko asmorik gabe. Gure proposamenak, (5.12)-k, ordea, egungo eta aurreko gradienteak berreskalatzen ditu iterazioz iterazio, g_t-k $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$ imita dezan —gogoratu 5.2. irudia—.

[4]Ingelesetik, *SGD with Momentum (SGDM)*.

(5.15). ekuazioak SGDM optimizatzailea gradiente zehatzeko interprete gisa berrinterpretatzen du, ikaskuntza-tasa berreskalatuz. Aldiz, gure memorian oinarritutako proposamenak gradientearen interpretazio zehatza eskaintzen du, ikaskuntza-tasa libre utziz. Ondorioz, ikaskuntza-tasa gradienteetan oinarritutako optimizatzailearen hiperparametro independente bat da, eta ez entrenamenduan gradienteak interpretatzeko elementu lagungarri bat. Gainera, gure optimizatzailea memorian oinarritutako galerarekin (5.11) paraleloan lan egiteko diseinatuta dago, entrenamenduan zehar funtzio objektiboa (normalean eskuragarri ez dagoena) hurbiltzeko.

6. Conclusions and Future Work

One of the main benefits of Artificial Neural Networks (ANNs) is their ability to approximate nearly any continuous function to a desired precision. This is achieved through an underlying non-linear parameterization, typically turning the approximation task into a highly non-convex optimization problem.

In this dissertation, we have used ANNs to numerically solve PDEs. Although the ANN-based framework provides greater approximation capacity compared to traditional methods, it also produces significant drawbacks that hamper the performance during experimentation. Among the most remarkable, we highlight the following two: (a) lack of efficient integration methods for ANNs, especially in the presence of singular solutions where Monte Carlo (MC) integration fails; and (b) poor optimizer performance, which often leads to fall in a local minimum whose distance to a global minimum is uncertain.

Below, we collect the most salient aspects of the novel proposals exposed throughout this dissertation and our visions for future work that aim at overcoming the aforementioned limitations.

6.1. The Deep Finite Element Method

We developed a deep-learning-based method to solve PDEs that mimics a FEM setup when applying mesh refinements. Thanks to the FEM-based formulation, the integration is exact: the approximated solution is a linear combination of pre-established basis functions and the ANN only optimizes the involved optimal coefficients. Moreover, thanks to the proposed dynamic architecture, we are able to start from a low-dimensional discretization scheme and step-by-step scale to higher-dimensional spaces (when applying refinements), taking advantage of the solutions found in the previous subspaces.

The method is capable of solving both parametric and non-parametric PDEs. While solving non-parametric problems with the DeepFEM is computationally inefficient (because, essentially, we are unable to beat a traditional linear solver), the execution times when solving parametric problems are comparable to non-parametric ones (i.e., the number of employed iterations is similar), which is where the true power of the method is shown. We developed and implemented the DeepFEM in one spatial dimension employing piecewise-linear approximations and uniform refinements. Extending it to more complicated 2D or 3D geometries, with adaptive meshes, and higher-order polynomials is straightforward but challenging in a TensorFlow 2 (TF2) implementation. We leave this as future work.

Since our DeepFEM architecture is partially explainable, we identified equivalences between different types of training (end-to-end vs. layer-by-layer), the degrees of freedom arising in the FEM (coefficients of the involved linear combination), and the considered preconditioners and norms. However, the lack of convexity of the loss with respect to the trainable parameters is critical and prevents us from obtaining a robust method.

6.2. The Deep Double Ritz Method

We studied the problem of residual minimization for solving PDEs using NNs. Using the dual norm definition as a maximum over the test space, we first obtained a min-max saddle-point problem. This method turned out to be numerically unstable due to a lack of Lipschitz continuity of the test maximizers. To overcome this, we rewrote a general variational problem as a minimization of a Ritz functional employing optimal test functions. To carry out this minimization while computing the optimal test functions over general problems, we proposed a Deep Double Ritz Method (D^{2}RM) that combines two nested Ritz optimization loops using ANNs. This novel method constructs local approximations of the trial-to-test operator to express the optimal test functions as dependent on the trial functions. By doing this, ANNs allowed us to easily combine the two nested Ritz problems in a way that is difficult to treat with traditional numerical methods.

We tested the D^{2}RM in several smooth and singular problems. Numerical results illustrate the advantages and limitations of the proposed method. Among the advantages, we encounter the good approximation capabilities of ANNs, the generality of the proposed method, and its application to different linear PDEs, variational formulations, and spatial dimensions. As main limitations, we faced the aforementioned two common difficulties shared by most ANN-based PDE solvers, namely, the lack of efficient integration and optimization. Other than that, numerical results are promising and supported by a solid mathematical framework at the continuum level.

In future work, we will investigate the behavior of optimal test functions to propose enhanced stopping criteria in the inner loop. In particular, we believe that a global approximation of the trial-to-test operator using ANNs would be helpful for this purpose. In parallel, we will investigate reformulations in terms of residuals in replacement of the optimal functions. We believe that its integral formulation involving an integrand that converges to zero is key for stabilizing the challenge of numerical integration when using ANNs.

6.3. Memory-based Monte Carlo integration

With the aim of improving integration accuracy when using ANNs (without incurring prohibitive computational costs), we proposed a memory-based iterative method that conveniently accumulates MC integral estimations in previous iterations when the convergence reaches an equilibrium phase. Appropriately conveying this memory behavior to the gradient computation, we found a reinterpretation of the momentum method in gradient descent optimization. However, this integration error improvement that makes the loss function resemble the objective function (which similarly occurs with the gradient of the loss towards the gradient of the objective function) is insufficient for enhancing convergence, as we are limited to the inherent lack of convexity of the ANN parameterization. In other words, we are able to overcome the integration challenge but not the optimization.

6.4. Ongoing research

Our following research steps involve exploiting the interpretation of the ANN as a family of finite-dimensional vector spaces (recall Appendix 2.A). In this way, we are able to formulate a suitable goal-oriented adaptivity scheme using ANNs and to find an efficient optimization framework inspired by ultraweak formulations for Variational Physics-Informed Neural Networks (VPINNs). These ongoing projects are tentatively titled "*GOANNs: A Goal-Oriented Adaptivity scheme for solving Partial Differential Equations using Neural Networks*" and "*UltraPINNs: Exploiting ultraweak implementations to boost the performance of Variational Physics Informed Neural Networks*", respectively.

7. Lorpenak*

Tesi honek ANNen bidez PDEak ebazteko gaia lantzen du. Hiru ekarpen berritzaile nagusi proposatzen ditu, doktorego-programan zehar garatutako jarduera sorta arrakastatsu baten ondorio direnak.

Jarraian jaso dira 2019ko urrian Euskal Herriko Unibertsitateko (UPV/EHU) Matematika eta Estatistika doktoregoko programan hasi eta egundaino garatutako jarduera eta ikerketa-egonaldi garrantzitsuenak.

7.1. Argitalpenak

Pareek berrikusitako eta argitaratutako lanak.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-Based Monte Carlo Integration for Solving Partial Differential Equations Using Neural Networks.* Mikyška, J., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds) Computational Science – ICCS 2023. Lecture Notes in Computer Science, Vol. 14074, 2023. Springer, Cham.
https://doi.org/10.1007/978-3-031-36021-3_51.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method (D^2RM) for solving Partial Differential Equations using Neural Networks.* Computer Methods in Applied Mechanics and Engineering (Q1, Goreneko %1), Vol. 405, 115892, 2023.
https://doi.org/10.1016/j.cma.2023.115892.

2022 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Computer Methods in Applied Mechanics and Engineering (Q1, Goreneko %1), Vol. 391, 114562, 2022.
https://doi.org/10.1016/j.cma.2021.114562.

*See Appendix D for the English version/Ikus D. eranskina ingelesezko bertsiorako.

7.2. Kongresuak

Nazioko eta nazioarteko konferentzietan egindako ekarpenak. Aurkezlea agertzen da azpimarratuta.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *The Deep Double Ritz Method: a Deep Learning Residual Minimization Method for solving Partial Differential Equations.* 2nd IACM Mechanistic Machine Learning and Digital Engineering for Computational Science and Technology Conference. University of Texas El Paso-n, AEB. 2023ko irailak 24-27.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-based Monte Carlo integration for solving Partial Differential Equations using Neural Networks.* XXXI COMCA – Congreso de Matemática Capricornio, Antofagasta, Txile. 2023ko abuztuak 2-4.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-based Monte Carlo integration for solving Partial Differential Equations using Neural Networks.* ICCS 2023 – 23rd International Conference on Computational Science. Praga, Txekiar Errepublika. 2023ko uztailak 3-5.

2023 Carlos Uriarte, David Pardo, Ignacio Muga. *Goal-Oriented Deep Ritz and Least-Squares methods.* ICCS 2023 – 23rd International Conference on Computational Science. Praga, Txekiar Errepublika. 2023ko uztailak 3-5.

2022 Carlos Uriarte, David Pardo, and Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* SIAM MDS22 – Conference on Mathematics of Data Science. San Diego, Kalifornia, AEB. 2023ko irailak 26-30.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *Solving Partial Differential Equations using Adversarial Neural Networks.* CMN 2022 – Congress on Numerical Methods in Engineering. Las Palmas Kanaria Handikoa, Espainia. 2022ko irailak 12-14.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *Adversarial Neural Networks for solving variationally formulated Partial Differential Equations.* WCCM/APCOM 2022 – 15th World Congress on Computational Mechanics and 8th Asian Pacific Congress on Computational Mechanics. Yokohama, Japonia. 2022ko uztailak 31 – abuztuak 5.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *An Adversarial Networks approach for solving Partial Differential Equations.* ICCS 2022 – 22nd International Conference on Computational Science. Londres, Erresuma Batua. 2022ko ekainak 21-23.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *A Generative Adversarial Networks approach for solving Partial Differential Equations.* ECCOMAS 2022 – 8th European Congress on Computational Methods in Applied Sciences and Engineering. Oslo, Noruega. 2022ko ekainak 5-9.

2021 Carlos Uriarte, Ángel J. Omella, David Pardo. *A Finite Element based Deep Learning solver for parametric PDEs.* ICCS 2021 – 21st International Conference on Computational Science. Krakovia, Polonia. 2021eko ekainak 16-18.

7.3. Ikastaroak, mintegiak eta tailerrak

Ikastaro, mintegi eta tailerretan egindako ekarpenak. Azpimarratuta agertzen da ekarpen nagusia egin duen pertsona. Kasuren batean azpimarratutakorik ez egoteak esan nahi du ekarpen nagusia egin duen pertsona zehatzik ez dagoela.

2023 Carlos Uriarte, David Pardo. *(Goal-Oriented) Deep Residual Minimization Methods.* Mintegia Faculty of Computer Science, Electronics, and Telecommunications of the AGH University of Science and Technology-n, Krakovia, Polonia. 2023ko uztailak 11.

2023 Tomás Teijeiro, Ángel J. Omella, Jamie M. Taylor, Carlos Uriarte, David Pardo. *Parametric PDEs using Deep Learning.* Ikastaroaren eta talde-tailerraren antolatzailea Asturiasen, Espainia. 2023ko maiatzak 21-27.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations using Neural Networks.* Ahozko aurkezpena Workshop Numerical Methods in Geophysics: Present, Future, and Applications-en, Valparaíso, Txile. 2023ko urtarrilak 12-13.

2023 Ángel J. Omella, Carlos Uriarte, and David Pardo. *Coding Deep Neural Networks for PDEs.* Irakaslea Pontificia Universidad Católica de Valparaíso-n antolatutako eta Olmué-n garatutako ikastaroan, Txile. 2023ko urtarrilak 15-20.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations.* Ahozko aurkezpena XC Encuentro Anual de la Sociedad Matemática de Chile Punta de Tralca-n, Txile. 2022ko abenduak 8-10.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations.* Mintegia Pontificia Universidad Católica de Valparaíso-ko Matematikako Institutan, Txile. 2022ko azaroak 11.

2022 David Pardo, Magdalena Strugaru, Jamie M. Taylor, Ángel J. Omella, Jon A. Rivera, Carlos Uriarte, Ignacio Muga, Judit Muñoz-Matute. *Deep Learning for Simulation and Inversion Problems.* Hitzaldi nagusia Oden Institute for Computational Engineering and Sciences of the University of Texas-en, Austin, Texas, AEB. 2022ko urriak 21.

2022 David Pardo, Ángel J. Omella, Jamie M. Taylor, Carlos Uriarte, Jon A. Rivera, Magdalena Strugaru. *Deep Learning for Simulation and Inversion Problems.* Hitzaldi nagusia 9º Congreso Metropolitano en Modelización y Simulación Numérica-n, Mexiko. 2022ko maiatzak 4-6.

2021 David Pardo, Ángel J. Omella, Jon Ander River, Carlos Uriarte, Ana Fernádez-Navamuel. *Solving Forward and Inverse Problems with Deep Learning.* Ikastaroaren eta talde-tailerraren organizatzailea Kantabrian, Espainia. 2021ko maiatzak 10-18.

2021 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Mintegia Pontificia Universidad Católica de Valparaíso-ko Matematikako Institutuan. 2021ko martxoak 26.

2021 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Mintegia Faculty of Computer Science, Electronics, and Telecommunications of the AGH University of Science and Technology-n, Krakovia, Polonia. 2021ko urtarrilak 21.

2020 David Pardo, Ángel J. Omella, Carlos Uriarte. *Deep Learning for Solving Inverse Problems using TF2.0.* Ikastaroaren eta talde-tailerraren antolatzailea Asturiasen, Espainia. 2020ko ekainak 28- uztailak 4.

7.4. Ikerketa-egonaldiak

Hogeita hamar egun baino gehiago iraun duten ikerketa-egonaldiak.

2023 Pontificia Universidad Católica de Valparaíso-ko Matematikako Institutuan, Txile. Kudeatzaile-akademikoa: Ignacio Muga irakaslea (92 egun).

2022 Oden Institute for Computational Engineering and Sciences of the University of Texas at Austin-en, AEB. Kudeatzaile-akademikoa: Leszek F. Demkowicz irakaslea (37 egun).

7.5. Dibulgazio-jarduerak

Dibulgazioko hitzaldiak.

2023 Elisabete Alberdi, Carlos Uriarte. *Matematika eguneroko bizitzan. Ekuazio diferentzialak mundua azaltzeko.* Dibulgazio hitzaldia Uhagon Kulturgunean, Markina-Xemein, Bizkaia, Espainia. 2023ko maiatzak 12.

2022 Carlos Uriarte. *Ekuazio diferentzialak mundua azaltzeko eta sare-neuronal artifizialak horiek ebazteko.* Dibulgazio hitzaldia Matematika Eguneroko Bizitzan 19. Edizioan, Bidebarrieta Kulturgunean, Bilbo, Espainia. 2022ko maiatzak 19.

Appendices/Eranskinak

A. Introduction (Chapter 1)

A.1. Motivation

Partial Differential Equations (PDEs) are of great value to society due to their broad applicability in modeling multiple biological, physical, or social phenomena [73, 207, 70, 192]. Using derivatives in these equations allows us to describe complex relationships and their rates of change over time and/or space. For example, they allow to model: heat transfer [102, 88], electromagnetic fields [106, 221], fluid dynamics [23, 14, 223], population evolution [96, 160, 38], and financial [26, 101] or health-care [193, 141, 165] forecasts.

However, establishing an equation to describe a physical or social situation is only the first step. There arise different kinds of problems when modeling via PDEs, and the way we treat or approach them depends on the available computational resources.

In *forward problems*, we are interested in determining the function that satisfies a PDE given some initial or boundary conditions. In this way, the solution to forward problems can provide insights into the modeled phenomena and can be used to make predictions, usually via post-processed measurements taken from the solution.

In *inverse problems*, we have measurements of the solution, but we need to discover the value of specific parameters in the equation. This is an indirect problem, as we need to determine the parameters that provide a PDE solution that agrees with the given measurements [210]. Example A.1 illustrates a possible framework of forward and inverse problems in electromagnetic fields [72].

Example A.1 (Electromagnetic fields)**.** Maxwell's equations model electromagnetic fields according to four well-known physical laws [72, 106]:

$$\begin{cases} \nabla \times \mathbf{E} = -j\omega\boldsymbol{\mu}\,\mathbf{H} - \mathbf{M}, & \text{Faraday's Law,} \\ \nabla \times \mathbf{H} = (\boldsymbol{\sigma} + j\omega\boldsymbol{\varepsilon})\mathbf{E} + \mathbf{J}, & \text{Amperè's Law,} \\ \nabla \cdot (\boldsymbol{\varepsilon}\mathbf{E}) = \rho_f, & \text{Gauss' Law of Electricity,} \\ \nabla \cdot (\boldsymbol{\mu}\mathbf{H}) = 0, & \text{Gauss' Law of Magnetism.} \end{cases} \tag{A.1}$$

Here, **E** and **H** denote the electric and magnetic fields in the frequency domain, respectively; **J** and **M** are given source terms; $\boldsymbol{\sigma}$ stands for the electrical conductivity of the media, $\boldsymbol{\varepsilon}$ for the electrical permittivity, $\boldsymbol{\mu}$ for the magnetic permeability, ρ_f for the electric charge density, j for the imaginary unit, and ω for the angular frequency. Then, a possible forward-inverse scenario on electromagnetic fields could consist of parameters $\{\boldsymbol{\sigma}, \boldsymbol{\mu}, \boldsymbol{\varepsilon}\}$ and measurements $\{\mathbf{Z}\}$, where $\mathbf{Z}$ denotes the impedance tensor defined by $\mathbf{E} = \mathbf{Z}\mathbf{H}$. For further details, see, e.g., [9]. Figure A.1 shows a graphic summary of this modeled framework.

$$\begin{tikzcd}
\begin{array}{c}\text{Parameters}\\ \{\boldsymbol{\sigma}, \boldsymbol{\mu}, \boldsymbol{\varepsilon}\}\end{array} \arrow[r, "\text{Solve PDE}"] \arrow[rr, bend left, "\text{Forward problem}"] & \begin{array}{c}\text{Solution}\\ \{\mathbf{E}, \mathbf{H}\}\end{array} \arrow[r, "\text{Post-processing}"] & \begin{array}{c}\text{Measurements}\\ \{\mathbf{Z}\}\end{array} \arrow[ll, bend left, "\text{Inverse problem}"]
\end{tikzcd}$$

Figure A.1: Forward and inverse problems sketch in electromagnetic fields.

Solving forward problems requires, in general, fewer resources than inverse problems. Given a choice of parameters, a forward problem typically consists of performing a single PDE simulation. In contrast, an inverse problem usually requires the iterative resolution of forward problems, conveniently readjusting the choice of parameters at each iteration based on the outcome of previous forward simulations [210, 103].

Connecting both types of problems, we have *parametric problems*, where the aim is to study the behavior of the solution/measurements as a function of the parameters. The solution can depend on one or several parameters, and we can study how changes in these parameters affect the behavior of the system. Alternatively, the parametric approach can be viewed as a forward problem where the parameters of the equation take the role of variables of the solution. Typically, the parametric forward problem is nonlinear with respect to the introduced new variables, enormously escalating and hindering its resolvability.

In this dissertation, we will only address the task of solving parametric and non-parametric linear PDEs, which is critical in many applications and frameworks. We will indistinctly call PDEs both one-dimensional—also known as *Ordinary Differential Equations (ODEs)*—and higher-dimensional differential equations.

A.2. Traditional Numerical Methods

There are various methods for solving PDEs, which can be broadly classified into two categories: analytic and numerical methods.

Analytic methods involve using mathematical techniques such as separation of variables, integral transforms, or complex analysis to find an exact solution to the PDE [205, 87, 213]. However, these methods are often limited to relatively simple PDEs and geometries. Additionally, even if the PDE is analytically solvable, the resulting solution may be expressed in a complicated form (e.g., as an infinite power series), which can be difficult to interpret and apply in practical scenarios.

On the other hand, numerical methods involve approximating the solution to the PDE using a computational algorithm. Generally, these methods propose approximations that are easily interpretable or numerically recoverable (e.g., as a finite linear combination of simple prescribed functions). Below, we briefly review the main aspects of three widely known and employed kinds of numerical methods for solving PDEs.

The *Finite Difference Method (FDM)* discretizes the spatial domain and/or time interval of the PDE into a finite number of subdomains [203, 178, 130]. Then, the evaluation of the derivatives at these discrete points is approximated by solving algebraic equations containing finite differences and values from nearby points. The FDM converts a PDE into a system of linear equations that can be solved by matrix algebra techniques. It is conceptually simple but challenging in design for complex problems or irregular domains. In particular, it suffers from the so-called *curse of dimensionality* [29, 30], where the size of the involved matrix grows exponentially with the dimension of the problem.

Similarly, the *Finite Element Method (FEM)* [100, 39, 138, 184] proposes a mesh-based scheme where the approximated solution is in the form of a finite linear combination of some prescribed functions (typically, piecewise polynomial) with local support. The support is purposely designed on specific regions of the domain, called elements, so the resulting matrix is sparse. The FEM is particularly suitable for dealing with PDEs in variational form but suffers from similar difficulties as the FDM.

Spectral methods [81, 36, 46] are a class of numerical methods that approximate functions using a linear combination of orthogonal basis functions with global support, such as Chebyshev polynomials or Fourier series. These methods can achieve very high accuracy, and are particularly useful for smooth solutions with high oscillatory behavior, but may become computationally unfeasible for complex geometries.

In essence, the numerical methods mentioned above are based on pre-establishing a finite-dimensional basis and parameterizing the approximation via the coefficients of the corresponding linear combination. When using a mesh-based basis (in the FDM and the FEM), it must be sufficiently fine to conveniently capture possible complex behaviors of the exact solution, but without falling into the computational disadvantage of over-refining. In spectral methods, the major challenge involves selecting orthogonal basis functions for arbitrary domains.

A.3. Neural Networks are Universal Approximators

Artificial Neural Networks (ANNs), or *Neural Networks (NNs)* for short, provide an alternative approximation approach to the traditional linear combination of prescribed functions.

The most basic form of NNs are *Feed-Forward Neural Networks (FFNNs)* [190, 105, 12, 77, 195], which are inspired by the structure and function of biological NNs in the brain. They consist of interconnected *neurons* that are organized into layers and the information flows in one direction, from one layer to the following one. Each neuron receives input from neurons in the previous layer and computes a weighted sum. When such sum is calculated from *all* neurons in the previous layer, we say that the FFNN is *fully connected*. This sum is then passed through a non-linear activation function to later transmit the outcome to the neurons in the following layer. We defer the formal description of fully-connected FFNN to Chapter 2.

Fully-connected FFNNs are universal approximators. Roughly speaking, this means that they are able to approximate almost any given function to a desired precision.

One of the first versions of the *Universal Approximation Theorem (UAT)* entails the *arbitrary-width* case, which states that a FFNN with a single hidden layer is a universal approximator of continuous functions as long as it is sufficiently wide. This result was parallelly and independently shown in 1989 for sigmoid activation functions [54], as well as for non-constant, bounded, and monotonically-increasing activation functions [98]. The principal author of the aforementioned last work showed two years later that it is not the specific choice of the activation function but the FFNN architecture itself that endows NNs the potential of being universal approximators [97]. Related to this, it was thereafter shown that the universal approximation property is equivalent to having a non-polynomial activation function [129, 175].

The "dual" version of the arbitrary-width case for the UAT consists of fixing a bounded width and establishing an *arbitrary depth*. Several authors have studied this reformulation since the early 2000s [83, 118], with a particular emphasis on ReLU activation functions [230, 140, 90, 89]. Analogously, we encounter results entailing *bounded-width* and *bounded-depth* assumptions. For example, [146] claims the existence of a sigmoidal-type analytic activation function for a FFNN with depth two and bounded width being a universal approximator. Additionally, [85] proved that FFNNs with a single hidden layer with bounded width are still universal approximators for univariate functions, which generally does not hold for multivariate functions.

Several extensions to the UAT have been developed over the past three decades dealing with discontinuous activation functions [129], non-compact domains [118], and alternative architectures and topologies [118, 136]. We specifically highlight the quantitive version of the UAT [22], which discusses the capability of FFNNs to be universal approximators overcoming the *curse of dimensionality* compared to classical piecewise-linear function spaces. For more extensive discussions about this, we refer to the works of [176, 19, 24, 226] and the references therein.

A proliferation of research initiatives and investigations have demonstrated through evidence and empirical tests that suitable combinations of different kinds of architectures, such as *convolutional*, *recurrent*, and *residual* NNs (CNNs, RNNs, and ResNet, respectively), often surpass the approximation and computation capabilities of FFNNs in different scenarios and applications [52, 59, 132, 206, 77, 93, 222, 4, 67, 220].

In this dissertation, we will limit ourselves to fully-connected FFNN architectures. Therefore, the terms NNs and FFNNs can be interchangeably read from now on unless otherwise specified. Moreover, we emphasize that the study of the approximation capacities of the considered architectures is beyond the scope of this dissertation. In other words, we will (naively) assume that our selected architectures during experimentation have a "sufficiently desired degree of approximation capacity".

A.4. Literature Review

In recent years, a great deal of work has emerged and focused on employing NNs to solve PDEs.

A.4.1. The first works solving PDEs using NNs

The very first documented work addressing the PDE-solving task following the terms in Section A.3 is dated to 1994[1] [63]. Although some earlier works already used NNs in various PDE-driven scenarios (see, e.g., [127, 216, 152]), proposal [63] was the earliest to exploit the representation of a PDE solution by means of a NN that is supported on the universal approximation property.

[1]Received by the publisher in August 1992, revised in August 1993, and first published in March 1994.

Such work proposes a FFNN architecture with sigmoid activation functions to approximate the solution of a PDE with boundary conditions—also called *Boundary-Value Problem (BVP)*—described in the form of

$$\begin{cases} Au = f, & \text{in } \Omega, \\ Du = g, & \text{in } \partial\Omega, \end{cases} \tag{A.2}$$

where Ω is a bounded open domain with boundary $\partial\Omega$, f is a given source function, and A and D are some given (differential/boundary) operators. The authors literally claimed[2]: "A so-called strong solution of (A.2) is an element of some underlying space $\mathbb{U}$ which satisfies both the PDE and the boundary conditions. [...] However, if there is a 'universal' approximator in this underlying space, that is, a function that has enough parameters and can be arbitrarily close to any element of the space by a judicious choice of the parameters, then this approximator can be a good choice for the approximate solution sought for. The closeness of two functions is described by the concept of denseness in $\mathbb{U}$. For a more mathematical definition, see Hornik et al. [98]".

Then, the PDE-solving task is stated as a minimization of the *objective function* given by

$$\mathcal{F}(u_{\mathrm{NN}}) = \int_{\Omega} \|Au_{\mathrm{NN}} - f\|^2 + \int_{\partial\Omega} \|Du_{\mathrm{NN}} - g\|^2, \tag{A.3}$$

where u_{NN} is the considered FFNN model. Originally, $\|\cdot\|$ was unspecified but probably thought of as the usual discrete 2-norm so that the minimization of $\mathcal{F}$ interprets within an L^2-minimization scheme (as suggested in [53]). After such presentation, the authors said[3]: "It is now possible to compute Au and Du, in closed form, in terms of x. We propose to use point collocation and enforce (A.3) at selected locations in Ω and $\partial\Omega$. The resulting objective function may then be minimized to find values for [the learnable parameters of the NN]—hence the approximate solution to (A.2)". They specified that the minimization of the objective function was carried out according to "a quasi-Newton method" and employing "finite-difference gradients" (see, e.g., [61] and [156] for related references).

Since then, several works have emerged within the PDE-solving framework using NNs. Below, we review some of the most relevant ones from the late 1990s to the early 2010s.

[2] Original symbols and references have been adapted for suitability with our presentation. [...] means that a portion of the original text has been omitted.

[3] As before, original symbols and references have been adapted for suitability. The portion of text between square brackets, [`text`], indicates that we inserted or replaced the original text for convenience in the understanding.

In 1994, [154, 153, 71] proposed the utilization of a FFNN with a single hidden layer activated by a “hard limit” function (so named by the authors) to solve linear and non-linear ODEs. This choice of activation function allows interpreting the set of admissible NNs as a family of piecewise-linear functions, and thus approximate the ODE solution in those terms. In 1998, [124] introduced a scheme to strongly impose the boundary conditions as a composition of a (boundary-free) FFNN with a filter, as opposed to [63], where the boundary conditions were specified “mildly” via the objective function. In 1999, [164] reported a NN implementation for the numerical approximation of functions of several variables and their partial derivatives accompanied by several results. Among those, we encounter applications to PDEs as part of a boundary element method for analyzing viscoelastic flows. In 2000, [94] proposed an extended backpropagation algorithm to solve a class of first-order PDEs with particular concern on nonlinear control systems, and [125] presented a NN-based model consisting of two different FFNN-type architectures to address PDEs with boundary conditions on irregular geometries.

Among the works from 2001 to the mid-2010s, we find multiple works which can be understood as continuations of those mentioned above. Specifically, [124] and [94] established significant precedents for the architecture designs and training setups, respectively, of the following works: [1] introduced a method combining NNs and evolutionary algorithms for solving PDEs and their boundary conditions; [202] employed NNs to analyze the dynamics of Kuramato-Sivashinsky and Navier-Stokes nonlinear equations; [147] presented a hybrid method for solving high-order ODEs; [199] addressed Schrödinger’s equation by a FFNN with an improved energy-function scheme via unsupervised training; [28] proposed a combination of minimization techniques and collocation methods via NNs to establish approximate closed-analytic-form solutions to PDEs; [215] used FFNNs in combination with grammatical evolution and local optimization to solve (systems of) ODEs and PDEs (see [214] for a precedent in grammatical-evolution-based neural-network training); and [151] presented a NN-based method for solving PDEs with irregular domains with mixed boundary conditions. Additionally, we encounter two major classes of extensions of NN-based PDE-solving frameworks: (Multiquadratic) Radial Basis Function (RBF) networks[4] [142, 143, 108, 144, 109, 113, 145, 76, 13, 75, 47, 121]; and NNs inspired in the FEM [31, 60, 234, 183, 148, 15, 110, 120]—sometimes also referred to as *Finite Element Neural Networks*. We refer to [227] for a detailed overview of many of the abovementioned works.

[4]See [170] for a UAT precedent on RBF networks.

A.4.2. Solving PDEs using NNs since TensorFlow

Prior to the first open-source release of *TensorFlow (TF)* [3, 2] in 2015, the absence of efficient *Automatic Differentiation (AD)* posed challenges in NN-based research. AD is nowadays crucial to compute gradients of NNs during training [82, 27, 150]. Without it, researchers and practitioners had to manually design the computation scheme of gradients, which typically was time-consuming and prone to errors, significantly as models grew in complexity. Consequently, innovation and experimentation were hampered. The inability to quickly try new ideas and different types of models slowed down progress, making it difficult for newcomers to enter the field.

With the introduction of TF [3, 2] in 2015 and of PyTorch [171, 172] in 2017, AD became more accessible for researchers, liberating them from manual computation designs and fueling faster development and innovation. Thanks to the emergence of these NN-oriented, AD-based, and user-friendly platforms, there arose a specific interest in the PDE-solving community to investigate NN-based venues in their research from the late 2010s on.

Specifically, *Physics-Informed Neural Networks (PINNs)* could be considered as one of the most significant precedents that have boosted the PDE-solving community to employ NNs in their works, especially during the last five years. The first version of PINNs was released as a two-part treatise in 2017 [180, 181]. There, the authors proposed data-driven alternatives for either solving (in [180]) or discovering (in [181]) PDEs. To distinguish the "solving" from the "discovering" frameworks, we think of a PDE with a differential operator governed by certain coefficients (parameters). On the one hand, if we know the values of all the coefficients, then we have complete access to the differential law involved in the PDE. Hence, we can reduce everything to the terms of the aforementioned "first work" [63][5] for approximating the PDE solution (recall the first part of Section A.4.1). On the other hand, if some of the coefficients are unknown, but we have access to some observations of the PDE solution at different spatio-temporal points, then we can establish a supervised learning scheme where a NN (representing the PDE solution) together with some additional trainable parameters (representing the unknown PDE coefficients) are trained constrained to satisfy both the PDE and the accessible labeled data.

Continuing with the PINN terminology, we encounter works addressing variational formulations, named *(hp-)variational PINNs ((hp-)VPINNs)* [115, 116,

[5]Interestingly, neither [180, 181] nor similar authorship reviews such as [182, 114] do cite [63]. We need to consult more general (and authorship-independent) reviews to find this relation. For example, [53] states: "The concept of incorporating prior knowledge into a machine learning algorithm is not entirely novel. In fact, Dissanayake and Phan-Thien [63] can be considered one of the first PINNs".

188]; conservation laws, named *conservative PINNs (cPINNs)* [107]; fractional-order PDEs, named *fractional PINNs (fPINNs)* [168]; and Bayesian proposals for noisy data, named *Bayesian PINNs (B-PINNs)* [229]. Among PINN applications to real-world phenomena, we highlight: [149] for high-speed flows, [157, 99] for power systems, [200] for ultrasound nondestructive quantification, [43] for heat transfer, and [42, 111] for fluid mechanics. We refer to [53, 126, 35] for more extensive reviews in PINNs.

Alternatively, related works adopting the *deep* first name have also been proposed following the NN-based PDE-solving aim. To mention a few: [68, 69] introduces the *Deep Ritz Method (DRM)*, which minimizes the Ritz energy functional involved in variational formulations of PDEs—refer to [139, 65, 135, 217] for related further analyses and extensions); [201] presents the *Deep Galerkin Method (DGM)*—refer to [49, 131, 7, 198] for further extensions, applications, and comparisons; [45] proposes a *Deep Least-Squares (DLS) method*—refer to [44, 137, 161] for further extensions; and [212] presents a *Deep Fourier Residual (DFR) method* with a recent extension to time-harmonic Maxwell's equations [211].

We also encounter several recent works regarding the resolution of parametric PDEs. Among the most relevant: [134] proposes a multi-level graph NN; [133] utilizes a parameterization in the Fourier space to achieve an expressive and efficient architecture suitable for parametric problems; [34] combines NNs with model reduction; [117] uses NNs to parameterize the physical quantity of interest as a function of input coefficients; [122] analyzes theoretically the approximability of parametric maps by NNs for parametric PDEs; and [173] extends PINNs to a parametric version throughout a named "metalearning" approach.

A.5. Major Contributions

This dissertation provides three major contributions to the resolution of PDEs using NNs: (i) a finite-element-based deep learning solver for parametric PDEs, (ii) a general residual minimization framework based on a double Ritz minimization scheme, and (iii) a numerical memory-based integration strategy for efficiently reducing the integration error during training.

A.5.1. The Deep Finite Element Method

Solving inverse problems is of great value to our society [228, 123] due to their broad applicability to multiple engineering disciplines such as imaging [33], electromagnetics [10], non-destructive evaluations [166], and geophysics [189], to mention a few. Different methods exist to solve them, including gradient-based and statistical-based methods [210, 16]. These traditional methods typically evaluate the inverse solution pointwise (i.e., for a given set of measurements) but rarely provide a global representation of an inverse operator. To overcome this and approximate a full inverse operator, it is possible to use NNs (see, e.g., [104, 231, 167, 196, 11, 197]) due to their universal approximation property.

Training NNs for solving inverse problems is typically performed in *Graphics Processing Units (GPUs)*, especially when dealing with large databases or complex model architectures. In these situations, it is convenient to have an efficient parametric PDE solver implemented within the GPU in the form of a NN to rapidly iterate over the different parameter candidates during the inverse problem training. Under these conditions, there are two main approaches for training the inverse model: (i) pre-training beforehand the parametric forward model to later use it as a real-time solver/evaluator, or (ii) training both the forward and inverse models simultaneously in an (auto) encoder-decoder scheme [77].

As the first main contribution of this dissertation, **we propose a NN-based model that acts as a parametric forward solver inspired by the FEM**. Specifically, our NN architecture is mesh-dependent, mimicking the effect of the dynamics when applying mesh refinements [18, 6, 208].

Our presentation and implementation of the proposed method, called the *Deep Finite Element Method (DeepFEM)*, restricts to *one-dimensional (1D)* problems using piecewise-linear approximations and uniform refinements. The extension to higher-dimensional problems, higher-order polynomial approximations, and/or adaptive meshes is presumably straightforward but goes beyond the scope of this work. We test the DeepFEM on positive-definite and indefinite problems with constant and piecewise-constant parameters. In general, experiments show good approximations; however, on some occasions, the optimizer and convexity limitations prevent us from obtaining high-accuracy solutions.

A.5.2. The Deep Double Ritz Method

From an abstract point of view, variational formulations [155] (on which, for example, the FEM is based) consist of translating BVPs into functional equations of the form

$$Bu = l \in \mathbb{V}', \tag{A.4}$$

where $\mathbb{U}$ and $\mathbb{V}$ are the so-called *trial* and *test* (normed) spaces, respectively; $\mathbb{V}'$ is the dual of $\mathbb{V}$; and $B : \mathbb{U} \longrightarrow \mathbb{V}'$ and $l \in \mathbb{V}'$ are the "variational analogs" of the original differential operator and source term, respectively. We refer to [58, p. 150, eqs. (2.1)–(2.3)] for details. Specifically, we highlight the claim: "the nature of variational problems lies in the fact that the corresponding operator takes values in a dual space".

Under this abstract formulation, *residual minimization* naturally arises as a reformulation of (A.4) as follows:

$$\min_{u\in\mathbb{U}} \|Bu - l\|_{\mathbb{V}'}, \tag{A.5}$$

where the use of the dual norm $\|\cdot\|_{\mathbb{V}'}$ is a must because $Bu - l$ takes values in the dual space $\mathbb{V}'$ [58, p. 151, eq. (2.5)].

Following the supremum definition of dual norms, we end up with a saddle-point (min-max[6]) problem over the trial and test spaces as follows:

$$\min_{u\in\mathbb{U}} \max_{v\in\mathbb{V}\setminus\{0\}} \frac{\langle Bu - l, v\rangle_{\mathbb{V}'\times\mathbb{V}}}{\|v\|_{\mathbb{V}}}. \tag{A.6}$$

Here, $\langle\cdot,\cdot\rangle_{\mathbb{V}'\times\mathbb{V}}$ stands for the duality pairing and $\|\cdot\|_{\mathbb{V}}$ for the norm in $\mathbb{V}$.

In the context of NNs, [232, 20] proposed performing residual minimization employing two NNs to represent the trial and test functions during the min-max optimization strategy (A.6). They called the resulting method *Weak Adversarial Networks (WANs)*. Although [232, 20] considered suboptimal norms during experimentation—they employed the L^2-norm instead of the proper H^1-norm, the major underlying drawback of such min-max approach relies on the fact that seeking the involved unitary test maximizer for each trial minimization iteration is numerically unstable (we will show this in Chapter 4).

To overcome the above instability issue, **we propose a general residual minimization method using NNs**, called the *Deep Double Ritz Method (D^{2}RM)*, that follows a similar spirit as in WANs and utilizes two NNs to nestedly iterate over trial and test functions, but relying on a stable Ritz minimization formulation [185]. Indeed, D^{2}RM can be interpreted either as a stabilized version of WANs or as a generalization of the DRM [69] to problems that are not necessarily symmetric and positive-definite (recall Section A.4.2).

Following the interrelation between WANs, the DRM, and the D^{2}RM, we test and compare these three methods on several diffusion and convection problems to show the generalizability and stability of the D^{2}RM (compared to the other two) up to the approximation properties of the considered NNs and the training capacity of the optimizers.

A.5.3. Memory-based Monte Carlo integration

Solving a BVP with NNs is typically described as a minimization of an objective function in the form of a definite integral that involves a NN in the integrand,

[6]In Hilbert spaces, which is where we will limit ourselves later on, the supremum is attained and therefore, it can indistinctly be replaced by a maximum.

e.g., recall (A.3) or (A.6). Thus, the quality of the numerical approximation or quadrature is critical. If we were to consider a fixed quadrature grid for numerical integration during the entire training, it is highly probable that the NN acquires a greedy behavior that produces undesired NN predictions [186] as a result of the well-known overfitting phenomenon [92, 51, 77].

A usual choice for avoiding overfitting consists of considering *Monte Carlo (MC) integration* [163, 128] due to its stochastic nature. Moreover, its mesh-free structure allows a straightforward scalability to complex geometries or high-dimensional domains. Unfortunately, MC integration commits an error of order $\mathcal{O}(1/\sqrt{N})$, where N is the number of integration points [163]. This low convergence rate of integration accuracy is typically overcome by sampling tens or hundreds of thousands of integration points (at each training iteration), drastically decreasing the training speed.

As the third main contribution of this dissertation, **we propose a memory-based MC integration scheme when using NNs for solving PDEs**. Here, the memory is aimed at taking advantage of integral estimations computed in previous iterations to contribute to the integral estimate at the current iteration. This scheme gradually decreases the integration error without incurring the computational overhead of dealing with disproportionate samples during training. Additionally, by conveniently translating this approach to the *Stochastic Gradient Descent (SGD)* optimizer, we find a favorable reinterpretation of the momentum method [177].

A.6. Outline

The remainder of this dissertation is structured as follows. Chapter 2 formally introduces and analyzes FFNNs. Chapters 3, 4 and 5 are devoted to the detailed development of the main contributions of the dissertation briefly motivated in Sections A.5.1, A.5.2 and A.5.3, respectively. Chapter 6 exposes the main conclusions of the dissertation and sketches the challenges to be addressed in future work. Finally, Chapter 7 describes the main achievements of this dissertation, including the published works, scientific contributions to conferences and workshops, and other relevant scientific activities developed during the PhD program.

Go to Chapter 2 for the English reading continuation/Joan 2. kapitulura ingelesezko irakurketarekin jarraitzeko.

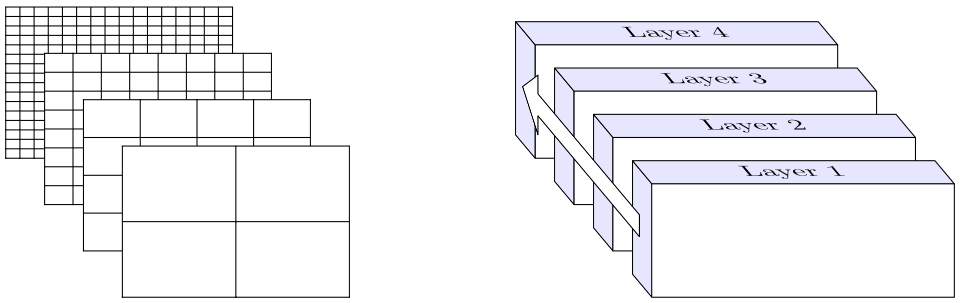
B. The Deep Finite Element Method (Chapter 3)

Summary. We introduce a dynamic Deep Learning architecture based on the Finite Element Method to solve linear parametric Partial Differential Equations. The connections between neurons in the architecture mimic the Finite Element connectivity graph when applying mesh refinements. We select and discuss several loss functions employing preconditioners and different norms to enhance convergence. For simplicity, we implement the resulting method in one spatial domain (1D), although its extension to 2D and 3D problems is straightforward. Extensive numerical experiments show, in general, good approximations for symmetric problems that are either positive-definite or indefinite in parametric and non-parametric equations. However, lack of convexity prevents us from obtaining high-accuracy solutions on some occasions. *Refer to [218] for the published version.*

B.1. Introduction

Many of the works discussed in the literature review (recall Section 1.4/A.4) address the idea of finding a continuous function (the NN) that approximates the solution of a (parametric) PDE. Generally, those designs allow evaluating the NN at any point in the domain, i.e., they have a mesh-free structure. However, they also present some limitations. We highlight the following two: (a) the resulting deep-learning-based architectures lack the so-called explainability [194, 21], and (b) numerical integration rules are challenging to design within the loss, as mentioned in [115] and further developed in [186].

To overcome both limitations, we propose a deep-learning-based method for solving parametric PDEs that resembles the FEM, called the *Deep Finite Element Method (DeepFEM)*. The proposed NN architecture aims to act as a solver of the parametric system of linear equations arising in the FEM and to mimic the Finite Element connectivity graph when applying mesh refinements—we associate each NN layer with a mesh refinement. Each NN layer has a ResNet [93] design and extends coarse solutions to finer meshes. Figure B.1 illustrates the relation between FEM refinements and NN layers. In this way, the architecture provides a certain degree of explainability, being the output the vector of nodal values corresponding to the finest mesh. In addition, this discrete approach enables exact numerical integration during training because the NN prediction lies on a piecewise-polynomial space.



Mesh refinements in the FEM   Layers in the DeepFEM

Figure B.1: Relation of mesh refinements in FEM with layers in the DeepFEM.

DeepFEM first sets an initial architecture that produces coarse solutions after training. Then, we iteratively and dynamically insert layers into the architecture, maintaining the previously trained learnable parameters and adding new ones. Subsequently, we retrain the learnable variables[1] of the new model and we repeat this process until we achieve a desired degree of accuracy in the solution. In this way, the proposed NN allows us to perform a mesh-convergence study.

For simplicity, our implementation restricts to 1D problems with piecewise-linear approximations and uniform refinements. The extension to higher dimensional problems, higher order polynomial approximations, and/or adaptive meshes is straightforward; but it requires a more elaborated implementation to deal with geometric criteria and node numbering between refinements, which we do not delve into in this work. We show numerical results of model problems with constant and piecewise-constant parameters.

The main contribution of the presented technology lies in the ability of the NN to solve parametric problems. For illustration, we first introduce it for the non-parametric case before considering the parametric problem. We do the same when describing the numerical results since the comprehensibility and limitations of the method for the non-parametric setting are extrapolable to the parametric one.

[1]Throughout this chapter, we change the terminology used in Chapter 2 from *parameters* to *variables*. We do so to distinguish the parameters associated with the PDE coefficients from those related to the weights and biases of the NN.

The remainder of this chapter is as follows. Section B.2 introduces our problem of interest and the corresponding variational formulation. Section B.3 describes our selected DeepFEM solver architecture and Section B.4 defines the loss when employing several preconditioners. Section B.5 shows implementation details and the limitations we encountered when using the library TF [3, 2]. Finally, Section B.6 discusses numerical results.

B.2. Model problem and Finite Element formulation

We focus on a particular parametric BVPs, although the presented approach applies to other problems that can be solved using the FEM.

Let Ω be a smooth domain and consider the following parametric linear BVP:

$$\begin{cases} -\nabla\cdot\sigma\nabla u + \alpha u = f, & \text{in } \Omega, \\ u = 0, & \text{in } \Gamma_D, \\ -\,\sigma\,\frac{\partial u}{\partial n} = g, & \text{in } \Gamma_N, \end{cases} \tag{B.1}$$

where parameters $\sigma > 0$ and $\alpha \in \mathbb{R}$ are piecewise-constant functions, f is the source, and g is the Neumann data. Γ_D and Γ_N are the Dirichlet and Neumann boundaries, respectively. n denotes the outer normal vector at each point of Γ_N and $\partial u/\partial n = \nabla u \cdot n$. Note that this model problem covers Poisson's ($\alpha = 0$) and Helmholtz's ($\alpha < 0; \sigma = 1$) equations.

A variational formulation of the above BVP reads as follows:

$$\begin{cases} \text{Find } u^* \in H_0^1(\Omega) \text{ such that} \\ (\sigma\nabla u^*, \nabla v)_\Omega + (\alpha u^*, v)_\Omega = (f, v)_\Omega - (g, v)_{\Gamma_N}, \forall v \in H_0^1(\Omega), \end{cases} \tag{B.2}$$

with

$$(u, v)_\Omega := \int_\Omega u \cdot v, \tag{B.3}$$

and where we have the following underlying spaces:

$$L^2(\Omega) = \{u : \Omega \longrightarrow \mathbb{R} : (u, u)_\Omega < \infty\}, \tag{B.4a}$$

$$H^1(\Omega) = \{u \in L^2(\Omega) : (\nabla u, \nabla u)_\Omega < \infty\}, \tag{B.4b}$$

$$H_0^1(\Omega) = \{u \in H^1(\Omega) : u|_{\Gamma_D} = 0\}. \tag{B.4c}$$

We assume that both f and g are sufficiently regular along the entire process, namely, $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$—we refer to [74] for a detailed discussion about boundary/fractional Sobolev spaces.

Following a FEM formulation in 1D, we look for a solution of the form

$$u_{\text{FEM}}(x; \sigma, \alpha) := \sum_{j=0}^{J} u_{\text{FEM},j}(\sigma, \alpha)\ \psi_j(x), \tag{B.5}$$

where $u_{\mathrm{FEM},j}(\sigma, \alpha)$ and $\psi_j(x)$ are the unknown coefficients and pre-established tent-shape piecewise-linear basis functions (see Figure B.2), respectively.

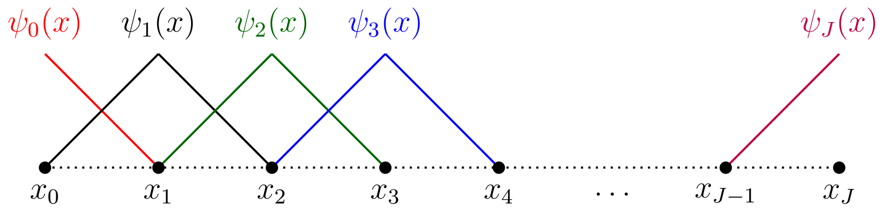


Figure B.2: Support of the tent-shape piecewise-linear functions ψ_j.

Inserting (B.5) in the variational formulation (B.2) and testing against all the basis functions ψ_j, we arrive at the following system of linear equations:

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{B.6}$$

where $\mathbf{A} = \mathbf{A}(\sigma, \alpha) := [(\sigma\psi_j', \psi_i')_\Omega + (\alpha\psi_j, \psi_i)_\Omega]_{i,j=0}^J$, $\mathbf{u} = \mathbf{u}_{\mathrm{FEM}}(\sigma, \alpha) := [u_{\mathrm{FEM},j}]_{j=0}^J$ is the vector of unknown coefficients, and $\mathbf{f} := [(f, \psi_j)_\Omega - (g, \psi_j)_{\Gamma_N}]_{j=0}^J$ is the load vector.

B.3. Dynamic architecture

We first describe our proposed architecture for the non-parametric case and then extend it to the parametric case. Finally, we consider both constant and piecewise-constant parameter alternatives.

Figure B.3 shows our selected node numbering when performing uniform mesh refinements. Accordingly, the extension operator $\mathbf{E}$ is given by a sparse matrix filled with ones and halves depending on the contribution with which each node propagates from the coarse to the fine mesh. Note that similar extension operators exist for 2D and 3D problems as well as for higher-order elements, and for $H(\mathrm{div})$, $H(\mathrm{curl})$, and L^2 discretizations [57].

B.3.1. Non-parametric scheme for constant PDE coefficients

Let σ and α be real-valued constant coefficients. For a one-element mesh, we propose the following architecture:

$$\mathbf{u}_{\mathrm{NN}}^{(1)}(\sigma, \alpha) = \mathbf{W}^{(1)}\ \varphi\left(\mathbf{W}_{\sigma,\alpha}^{(1)} \begin{bmatrix} \sigma \\ \alpha \end{bmatrix} + \mathbf{b}_{\sigma,\alpha}^{(1)}\right) \in \mathbb{R}^2, \tag{B.7}$$

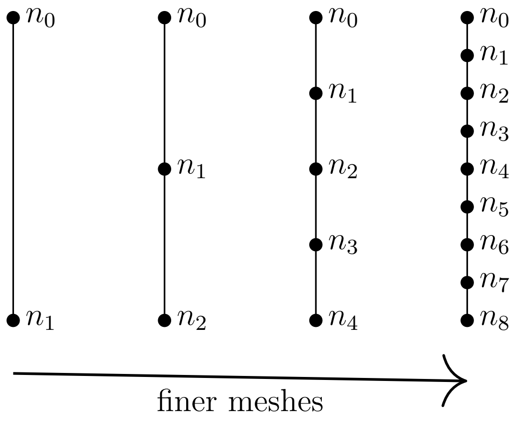


Figure B.3: Considered node numbering criteria for uniform mesh refinements in 1D.

where φ is an activation function, and the output is a vector of size two that aims to approximate the FEM solution in the considered (single-element) coarse mesh (step $s = 1$), i.e.,

$$\mathbf{u}^{(1)}_{\text{FEM}}(\sigma, \alpha) = \begin{bmatrix} u^{(1)}_{\text{FEM},0}(\sigma, \alpha) \\ u^{(1)}_{\text{FEM},1}(\sigma, \alpha) \end{bmatrix} \approx \begin{bmatrix} u^{(1)}_{\text{NN},0}(\sigma, \alpha) \\ u^{(1)}_{\text{NN},1}(\sigma, \alpha) \end{bmatrix} = \mathbf{u}^{(1)}_{\text{NN}}(\sigma, \alpha). \tag{B.8}$$

The set of learnable variables consists of

$$\theta^{(1)} = \{\mathbf{W}^{(1)}_{\sigma,\alpha}, \mathbf{b}^{(1)}_{\sigma,\alpha}, \mathbf{W}^{(1)}\}. \tag{B.9}$$

Then, we refine and obtain a two-element mesh (step $s = 2$). Its counterpart in the architecture consists of adding an input-dependent ResNet [93] to $\mathbf{u}^{(1)}_{\text{NN}}$ to define $\mathbf{u}^{(2)}_{\text{NN}}$ as follows:

$$\mathbf{r}^{(2)}_{\text{NN}}(\sigma, \alpha) = \mathbf{W}^{(2)}\, \varphi\left(\mathbf{W}^{(2)}_{\sigma,\alpha} \begin{bmatrix} \sigma \\ \alpha \end{bmatrix} + \mathbf{b}^{(2)}_{\sigma,\alpha}\right) \in \mathbb{R}^3, \tag{B.10a}$$

$$\mathbf{u}^{(2)}_{\text{NN}}(\sigma, \alpha) = \mathbf{E}^2_1 \mathbf{u}^{(1)}_{\text{NN}}(\sigma, \alpha) + \mathbf{r}^{(2)}_{\text{NN}}(\sigma, \alpha) \in \mathbb{R}^3, \tag{B.10b}$$

with $\mathbf{E}^2_1$ denoting the extension matrix of $\mathbf{u}^{(1)}_{\text{NN}}$ on the fine (two-element) mesh. The set of learnable variables $\theta^{(2)}$ is now

$$\{\mathbf{W}^{(2)}_{\sigma,\alpha}, \mathbf{b}^{(2)}_{\sigma,\alpha}, \mathbf{W}^{(2)}\} \qquad \text{or} \qquad \theta^{(1)} \cup \{\mathbf{W}^{(2)}_{\sigma,\alpha}, \mathbf{b}^{(2)}_{\sigma,\alpha}, \mathbf{W}^{(2)}\} \tag{B.11}$$

depending on whether we perform a *layer-by-layer* or an *end-to-end* training, respectively. Note that the new inserted learnable variables correspond only to

$\mathbf{r}_{\text{NN}}^{(2)}$. We initialize the variables of $\mathbf{r}_{\text{NN}}^{(2)}$ so that it is zero at the beginning of the retraining (e.g., initializing $\mathbf{W}^{(2)}, \mathbf{b}^{(2)} = 0$), and we maintain the learned values in $\theta^{(1)}$ in the previous step at the beginning of the retraining at (the current) step $s = 2$. We retrain $\mathbf{u}_{\text{NN}}^{(2)}$ (via layer-by-layer or end-to-end training regimes) so it approximates the FEM solution on the (two-element) fine mesh, i.e.,

$$\mathbf{u}_{\text{FEM}}^{(2)}(\sigma, \alpha) = \begin{bmatrix} u_{\text{FEM},0}^{(2)}(\sigma, \alpha) \\ u_{\text{FEM},1}^{(2)}(\sigma, \alpha) \\ u_{\text{FEM},2}^{(2)}(\sigma, \alpha) \end{bmatrix} \approx \begin{bmatrix} u_{\text{NN},0}^{(2)}(\sigma, \alpha) \\ u_{\text{NN},1}^{(2)}(\sigma, \alpha) \\ u_{\text{NN},2}^{(2)}(\sigma, \alpha) \end{bmatrix} = \mathbf{u}_{\text{NN}}^{(2)}(\sigma, \alpha). \tag{B.12}$$

We repeat this process iteratively (step by step), increasing the depth of our NN architecture until the number of elements is sufficient to accurately approximate the exact solution. Figure B.4 sketches this dynamic architecture.

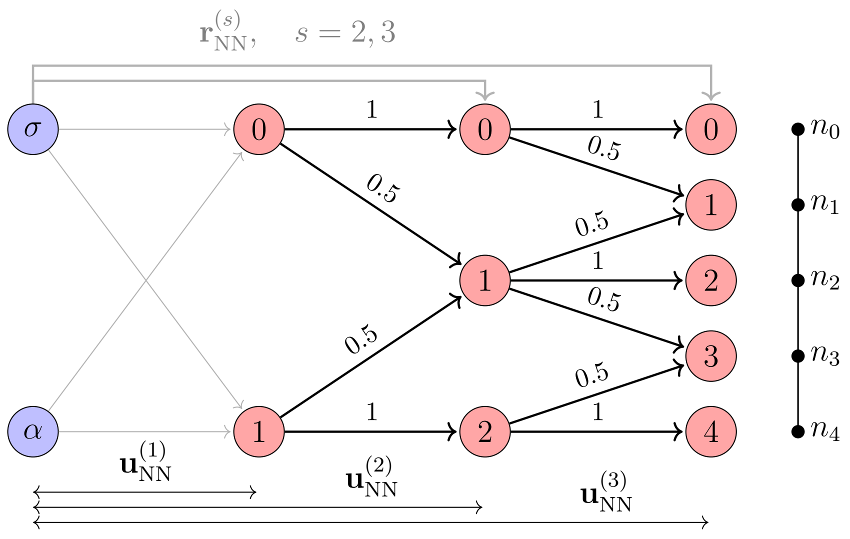

$\mathbf{r}_{\text{NN}}^{(s)}, \quad s = 2, 3$
σ
α
0
1
0
1
2
0
1
2
3
4
1
0.5
0.5
1
1
0.5
0.5
1
0.5
0.5
1
n_0
n_1
n_2
n_3
n_4
$\mathbf{u}_{\text{NN}}^{(1)}$
$\mathbf{u}_{\text{NN}}^{(2)}$
$\mathbf{u}_{\text{NN}}^{(3)}$

Figure B.4: Sketch of a non-parametric DeepFEM dynamic architecture. Gray arrows within the architecture indicate learnable variables, while black arrows refer to (non-trainable) extension operations.

B.3.2. Parametric scheme for constant PDE coefficients

The non-parametric scheme consisted of setting weights and biases of very low dimensionality (e.g.,$\mathbf{W}^{(1)}, \mathbf{W}_{\sigma,\alpha}^{(1)} \in \mathbb{R}^{2\times 2}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{3\times 2}, \mathbf{W}_{\sigma,\alpha}^{(2)} \in \mathbb{R}^{2\times 2}$) because we were only interested in training over a single sample of coefficients.

For the parametric problem (i.e., to train over data of coefficient samples), we add width and depth to the trainable pieces of the dynamic architecture as follows:

$$\mathbf{u}_{\text{NN}}^{(1)} = \boldsymbol{\mathcal{FC}}_{\text{NN}}^{(1)}(\sigma, \alpha), \tag{B.13a}$$
$$\mathbf{r}_{\text{NN}}^{(s)} = \boldsymbol{\mathcal{FC}}_{\text{NN}}^{(s)}(\sigma, \alpha), \qquad s \geq 2, \tag{B.13b}$$
$$\mathbf{u}_{\text{NN}}^{(s)} = \mathbf{E}_{s-1}^{(s)} \mathbf{u}_{\text{NN}}^{(s-1)} + \mathbf{r}_{\text{NN}}^{(s)}, \qquad s \geq 2. \tag{B.13c}$$

Here, $\boldsymbol{\mathcal{FC}}_{\text{NN}}^{(s)}$ denotes a fully-connected FFNN with non-activated last layer (recall Section 2.1) and whose output dimension is specified to match the dimensionality of the underlying FEM discretization. For simplicity, we call *trainable blocks* to the $\boldsymbol{\mathcal{FC}}_{\text{NN}}^{(s)}$ models. Figure B.5 shows the architecture of the trainable blocks and Figure B.6 shows them inside the parametric DeepFEM.

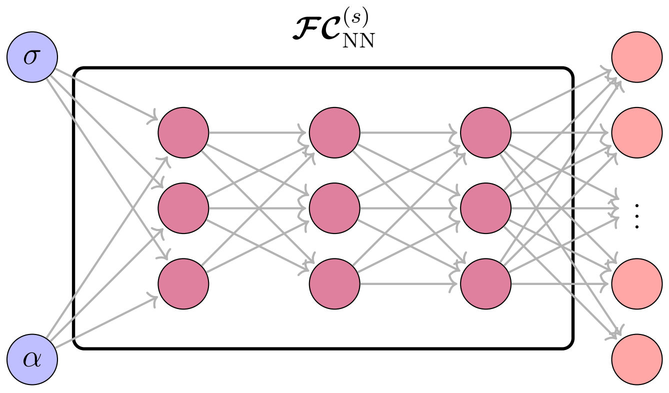


Figure B.5: Sketch of the architecture of a trainable block at step s. Gray arrows indicate learnable variables.

B.3.3. Parametric scheme for piecewise-constant PDE coefficients

Finally, we add the piecewise-constant behavior to the parameters that we assume taking constant values on each element of the initial mesh. Hence, the architecture is modified only in the input layer by replacing the σ and α values with vectors of sizes equal to the number of elements of the initial mesh. Figure B.7 illustrates this.

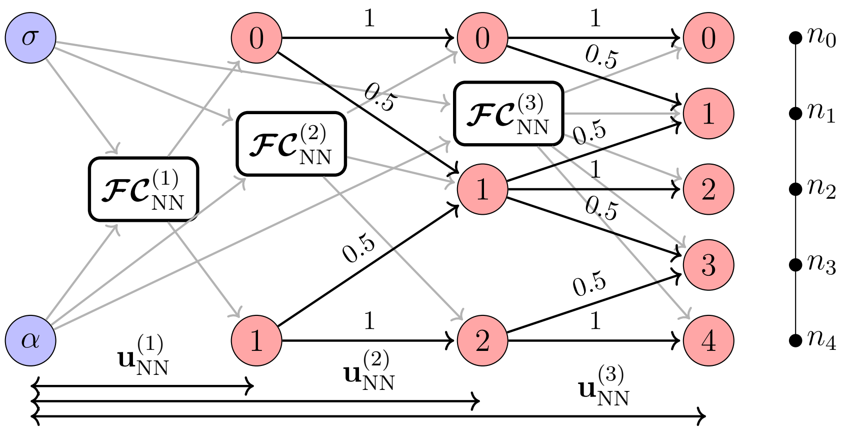

σ
α
$\mathcal{FC}_{\mathrm{NN}}^{(1)}$
$\mathcal{FC}_{\mathrm{NN}}^{(2)}$
$\mathcal{FC}_{\mathrm{NN}}^{(3)}$
0
1
0
1
2
0
1
2
3
4
1
0.5
0.5
1
0.5
1
0.5
0.5
0.5
1
n_0
n_1
n_2
n_3
n_4
$\mathbf{u}_{\mathrm{NN}}^{(1)}$
$\mathbf{u}_{\mathrm{NN}}^{(2)}$
$\mathbf{u}_{\mathrm{NN}}^{(3)}$

Figure B.6: Sketch of a parametric DeepFEM dynamic architecture. Gray arrows within the architecture indicate learnable variables, while black arrows refer to (non-trainable) extension operations.

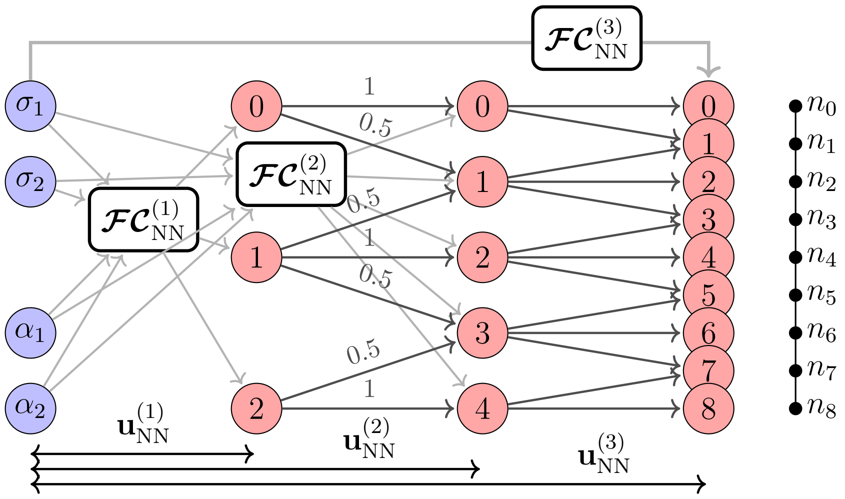

σ_1
σ_2
α_1
α_2
$\mathcal{FC}_{\mathrm{NN}}^{(1)}$
$\mathcal{FC}_{\mathrm{NN}}^{(2)}$
$\mathcal{FC}_{\mathrm{NN}}^{(3)}$
0
1
2
0
1
2
3
4
0
1
2
3
4
5
6
7
8
1
0.5
0.5
1
0.5
0.5
1
n_0
n_1
n_2
n_3
n_4
n_5
n_6
n_7
n_8
$\mathbf{u}_{\mathrm{NN}}^{(1)}$
$\mathbf{u}_{\mathrm{NN}}^{(2)}$
$\mathbf{u}_{\mathrm{NN}}^{(3)}$

Figure B.7: Sketch of a parametric DeepFEM dynamic architecture for piecewise-constant PDE coefficients. Gray arrows within the architecture indicate learnable variables, while black arrows refer to (non-trainable) extension operations.

B.4. Loss function and training

Within the step-by-step methodology described in Section B.3, we drop the superscript "(s)" for simplicity, although the learnable variables, loss functions, and considered norms are step-dependent.

To make the NN approximate the solution to the parametric system of linear equations arising in the FEM, i.e.,

$$\mathbf{u}_{\text{NN}} \approx \mathbf{u}_{\text{FEM}}, \tag{B.14}$$

we consider the loss function given by

$$\mathcal{L}(\theta, \{\sigma_i, \alpha_i\}_{i=1}^N) := \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{A}(\sigma_i, \alpha_i)\mathbf{u}_{\text{NN}}(\sigma_i, \alpha_i; \theta) - \mathbf{f}\|, \tag{B.15}$$

where $\mathbf{u}_{\text{NN}}(\sigma_i, \alpha_i; \theta)$ is the prediction of the DeepFEM model for the i^{th} sample, $\mathbf{A}(\sigma_i, \alpha_i)$ is the corresponding FEM matrix—recall (B.6), $\mathbf{A}(\sigma_i, \alpha_i)\mathbf{u}_{\text{NN}}(\sigma_i, \alpha_i; \theta) - \mathbf{f}$ is the residual vector, and $\|\cdot\|$ is a pre-established vector norm (e.g., the 2-norm).

B.4.1. Gradient-based training

To minimize the loss function, at the beginning of each step, we apply the Adam optimizer [119]. When the number of iterations attains an established maximum, we stop its execution. Alternatively, to control stagnation, we monitor and compare the value of the loss function obtained at any given iteration with the lowest loss obtained few iterations before. When the loss improvement is insignificant after a prescribed number of iterations, we stop the execution. Moreover, since the loss may increase or decrease during the Adam performance, we monitor the loss so as to return the best configuration of trainable variables at the end of its execution. Thereafter, we apply a customized gradient-descent-based optimizer with a loss-dependent adaptive learning rate (called *Adalr*). We maintain the same stopping and stagnation criteria as in Adam. The Adalr optimizer readjusts the trainable variables to prevent a loss increase. Algorithm 7 shows the learning rate adaptivity and trainable variables acceptance-rejection criteria. In the numerical results, we will compare in more detail both optimizers. As a general overview: Adam rapidly decreases the loss initially but tends to plateau at a suboptimal level, causing oscillations without significant improvement. On the other hand, Adalr monotonically reduces the loss, usually with a slower convergence speed compared to Adam.

Algorithm 7: Adalr optimizer

```
/* Superscripts denote training iterations.                      */
Input: $\theta^{(0)}, \eta^{(0)}, D$ ;    // Initial variables, learning rate, data
Output: $\theta^*$, $\mathcal{L}^*$ ;             // Final variables and loss values
$\mathcal{L}^{(0)} = \mathcal{L}(\theta^{(0)}; D);\ \theta^{(1)} = \theta^{(0)} - \lambda^{(0)} \frac{\partial \mathcal{L}}{\partial \theta}(\theta^{(0)}; D);$
$\theta^* = \theta^{(0)};\ \mathcal{L}^* = \mathcal{L}^{(0)};\ t = 1;$
while not STOP do
    $\mathcal{L}^{(t)} = \mathcal{L}(\theta^{(t)}; D);\ \theta^{(t+1)} = \theta^{(t)} - \lambda^{(t)} \frac{\partial \mathcal{L}}{\partial \theta}(\theta^{(t)}; D)$ ;
    /* If the new loss is worse                                  */
    if $\mathcal{L}^{(t)} > \mathcal{L}^*$ then
        $\lambda^{(t+1)} = \texttt{Decrease}(\lambda^{(t)});\ \theta^{(t+1)} = \theta^*;$
    else
        $\theta^* = \theta^{(t)};$
        /* If convergence is slow                                */
        if convergence is slow and no variables rejection in $t - 1$ then
            $\lambda^{(t+1)} = \texttt{Increase}(\lambda^{(t)});$
        else
            $\lambda^{(t+1)} = \lambda^{(t)};$
        $\mathcal{L}^* = \mathcal{L}^{(t)};$
    $t = t + 1;$
return $\theta^*$, $\mathcal{L}^*$
```

B.4.2. Norm selection and preconditioning

We want to select a discrete norm in the loss function—recall (B.15)—so that its behavior is similar to the error in the *energy norm*. In a symmetric and positive-definite problem, the energy norm is given by

$$\|\mathbf{v}\|_{\mathbf{A}} := \sqrt{\mathbf{v}^T\mathbf{A}\mathbf{v}}, \tag{B.16}$$

where $\mathbf{v}^T$ stands for the transpose of the column vector $\mathbf{v}$, and $\mathbf{A}$ denotes the symmetric and positive-definite matrix of the system of linear equations. For each (σ, α) sample, by writing the error vector as

$$\mathbf{e}(\sigma,\alpha) := \mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{A}^{-1}(\sigma,\alpha)\mathbf{f} = \mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{u}_{\text{FEM}}(\sigma,\alpha), \tag{B.17}$$

we arrive at the following norm relation with the residual:

$$\|\mathbf{e}(\sigma,\alpha)\|_{\mathbf{A}(\sigma,\alpha)} = \|\mathbf{A}(\sigma,\alpha)\mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{f}\|_{\mathbf{A}^{-1}(\sigma,\alpha)}. \tag{B.18}$$

In practice, the inverse of $\mathbf{A}(\sigma,\alpha)$ is non-computable and thus the error is never known. Hence, as in iterative methods for solving a system of linear equations, we consider preconditioners to mimic the inverse operator and thus decrease the condition number of the system [169]. Thus, we define the discrete norm $\|\cdot\|$ in (B.15) as

$$\|\mathbf{v}\| := \|\mathbf{v}\|_{\mathbf{P}} = \sqrt{\mathbf{v}^T\mathbf{P}\mathbf{v}}, \tag{B.19}$$

where $\mathbf{P} = \mathbf{P}(\sigma,\alpha)$ is a preconditioner for $\mathbf{A}(\sigma,\alpha)$. In particular, we consider block-Jacobi preconditioners of different block sizes and with one-element overlap [169]. Note that when $\mathbf{P} = \mathbf{I}$ is the identity matrix, the corresponding norm is the discrete 2-norm, i.e., $\|\cdot\|_{\mathbf{I}} = \|\cdot\|_2$.

In case the problem is indefinite, we consider a positive definite operator and corresponding matrix $\mathbf{B}$. The relation between the error and the residual in this new norm is given by

$$\|\mathbf{e}(\sigma,\alpha)\|_{\mathbf{B}} = \|\mathbf{A}^{-1}(\sigma,\alpha)\left\{\mathbf{A}(\sigma,\alpha)\mathbf{u}_{\text{NN}}(\sigma,\alpha) - \mathbf{f}\right\}\|_{\mathbf{B}}. \tag{B.20}$$

Following the above reasoning leads us to consider, in this occasion, the norm $\|\cdot\|$ in (B.15) as

$$\|\mathbf{v}\| := \|\mathbf{P}\mathbf{v}\|_{\mathbf{B}} = \sqrt{\mathbf{v}^T\mathbf{P}^T\mathbf{B}\mathbf{P}\mathbf{v}} = \|\mathbf{v}\|_{\mathbf{P}^T\mathbf{B}\mathbf{P}}. \tag{B.21}$$

Again, when $\mathbf{P} = \mathbf{B} = \mathbf{I}$ are the identity operators, the above equation reduces to the discrete 2-norm of the residual.

In the numerical results, in addition to using the energy norm for positive definite problems, we employ the L^2 and H^1 norms for testing and monitoring the (preconditioned) residual and the error. These continuum-level norms have their discretized counterparts as follows:

$$\|u\|_{L^2} = \sqrt{\mathbf{u}^T\mathbf{M}\mathbf{u}} = \|\mathbf{u}\|_{\mathbf{M}}, \tag{B.22}$$

$$\|u\|_{H^1} = \sqrt{\mathbf{u}^T\mathbf{M}\mathbf{u} + \mathbf{u}^T\mathbf{K}\mathbf{u}} = \|\mathbf{u}\|_{\mathbf{K}+\mathbf{M}}, \tag{B.23}$$

where $\mathbf{u} = [u_j]_{j=0}^J$ is the vector of evaluations at the nodal points of the mesh, ψ_j is the basis function related to the j^{th} node, and $\mathbf{M}$ and $\mathbf{K}$ are the mass and stiffness matrices defined by $(\psi_s, \psi_r)_\Omega$ and $(\psi_s', \psi_r')_\Omega$ in the $(r, s)^{\text{th}}$ entry, respectively. For convenience, we maintain the naming of the discrete norms (B.22) and (B.23) by the names inherited at their continuum level, namely, L^2 and H^1 norms, respectively.

Similarly, we drop the boldface notation when referring to the scalar-valued functions of the corresponding vector functions. For the NN predictions and FEM solutions,

$$u_{\text{NN}}(x; \sigma, \alpha) = \sum_{j=0}^{J} u_{\text{NN},j}(\sigma, \alpha)\ \psi_j(x), \tag{B.24a}$$

$$u_{\text{FEM}}(x; \sigma, \alpha) = \sum_{j=0}^{J} u_{\text{FEM},j}(\sigma, \alpha)\ \psi_j(x), \tag{B.24b}$$

where $u_{\text{NN},j}(\sigma, \alpha)$ and $u_{\text{FEM},j}(\sigma, \alpha)$ are the j^{th} components of the DeepFEM prediction and FEM solution vectors, $\mathbf{u}_{\text{NN}}(\sigma, \alpha)$ and $\mathbf{u}_{\text{FEM}}(\sigma, \alpha)$, respectively.

B.5. Implementation

We implement the described framework in the Python programming language, and we used the library TF2 [3, 2], NumPy [91], and SciPy [224] to build the NN models and to generate and manage the FEM data. We create the layers by redefining the corresponding base classes in Keras inside TF2 (`tf.keras`)[2]. We use a dedicated sparse library within TF2 (`tf.sparse`) to handle the extension operations and the FEM matrices to save memory and achieve high performance. We use double precision (float64) instead of the default single precision (float32). We calculate the loss function in an auxiliary non-trainable layer applied after the main model $\mathbf{u}_{\text{NN}}$. Below, we describe the three main difficulties encountered during our implementation.

[2]During the TF1.X era, Keras was an external library dedicated to facilitating the use of TF in Python. Since the TF2 release in September 2019, Keras became a sublibrary of TF2 until version TF2.12 in March 2023, where Keras became (again) an external framework built on top of TF that additionally supports JAX and PyTorch. See https://keras.io for further details (last accessed: September 4, 2023).

B.5.1. Reshape of the batch sparse tensors

Tensors flowing through Keras are prepared to maintain their first dimension (a.k.a axis) for the batch of samples. Consequently, matrix-vector multiplications in the loss function transform into an operation between a 3D sparse tensor (batch of sparse matrices) and a 2D tensor (batch of vectors) in the Keras model. We could define our operation via Einstein summation if both tensors were dense. However, there is still not an equivalent TF2 function supporting sparse tensors[3]. We overcome this by reshaping the 3D sparse tensor as a sparse non-overlapping block 2D matrix, and the 2D tensor as a long 1D vector. This leads us to miss the batch flow behavior of the model and be forced to avoid using the Keras training function. We thus perform a low-level optimization via AD in *eager execution*, which is significantly slower than training via *graph execution* within Keras[4].

B.5.2. FEM data generation

We utilize SciPy as the FEM environment to build the extension operators and the matrices and preconditioners of the system of linear equations. We also use it to solve the sparse systems via its built-in solver to compare model predictions. For more complicated geometries, there exist other more efficient software platforms to assemble the matrices, preconditioners, and extension operators (e.g., FEniCS [8]). We recommend their use for more complex FE systems (e.g., in 2D and 3D problems). In any case, these offline operations are calculated prior to the training of the NNs to not affect the optimization time of the model.

B.5.3. Preconditioners assembly and action

Ideally, we should manage preconditioners as LU block decompositions of the matrices of the system, and calculate their actions at each loss evaluation using a forward-backward substitution algorithm [5, 158]. Again, TF2 lacks an equivalent built-in function to evaluate these actions with sparse tensors. For this reason, we manage the preconditioners as already assembled tensors.

[3]We opened an issue concerning this regard in September 2020 in Github (`https://github.com/tensorflow/tensorflow/issues/43497`); however, we did not receive any robust support response yet (last accessed: September 4, 2023).

[4]See `https://www.tensorflow.org/guide/intro_to_graphs` for further details (last accessed: September 4, 2023).

B.6. Numerical results

We conduct a series of experiments to analyze the DeepFEM performance. Section B.6.1, Section B.6.2, and Section B.6.3 contemplate non-parametric problems, while Section B.6.4 addresses parametric problems. All experiments are carried out in the spatial domain $(0, 1)$ with Dirichlet and Neumann boundary conditions at 0 and 1, respectively.

B.6.1. A single PDE example

Let

$$\begin{cases} -u'' = -20x^3, \\ u(0) = 0, u'(1) = 5. \end{cases} \tag{B.25}$$

Its exact solution is $u^*(x) = x^5$. We solve it employing the DeepFEM starting from a one-element mesh ($s = 1$). We perform ten mesh refinements (eleven steps, i.e., $1 \leq s \leq 11$) to finish with a 1024-element overkill mesh. We analyze our proposed method in this first experiment when performing multiple refinements. All the training blocks of the model consist of one-neuron width and one-layer depth (recall Appendix B.3.2) employing ReLU activation functions on all trainable blocks.

Figure B.8 shows the predictions of DeepFEM over the first four steps before and after training. At each step, we observe that the model extends the coarse prediction to the fine mesh, and then (re)trains the resulting NN to obtain a proper fine grid approximation. Results show superb accuracy.

Figure B.9a shows the loss function evolution along the first four steps when it coincides with the energy-norm error, while Figure B.9b shows all eleven steps of the training process. This example illustrates the convergence behavior and limitations of DL under optimal conditions, i.e., when the inverse of the matrix is already part of the loss. The vertical jumps observed in the loss function correspond to the extension from one grid to the next one. After each jump, we observe a noisy convergence of the loss, which corresponds to the use of the Adam optimizer that works as an initial aggressive loss descender that gets stunned quickly. We then switch to the Adalr optimizer, which exhibits a monotonic loss decrease. At each phase, we select an initial learning rate equal to the first loss evaluation multiplied by 10^{-3} (for Adam) or by 10^{-2} (for Adalr). We set a maximum of 2,000 (Adam) and 4,000 (Adalr) iterations for each optimizer performance at each step. Moreover, if we attain a loss below 10^{-12}, we stop the optimization at each step. We carry out an end-to-end training of the DeepFEM model. We observe a convergence deterioration as we increase the step number (and grid size) that remains below 10^{-8} in all occasions (see Figure B.9c).

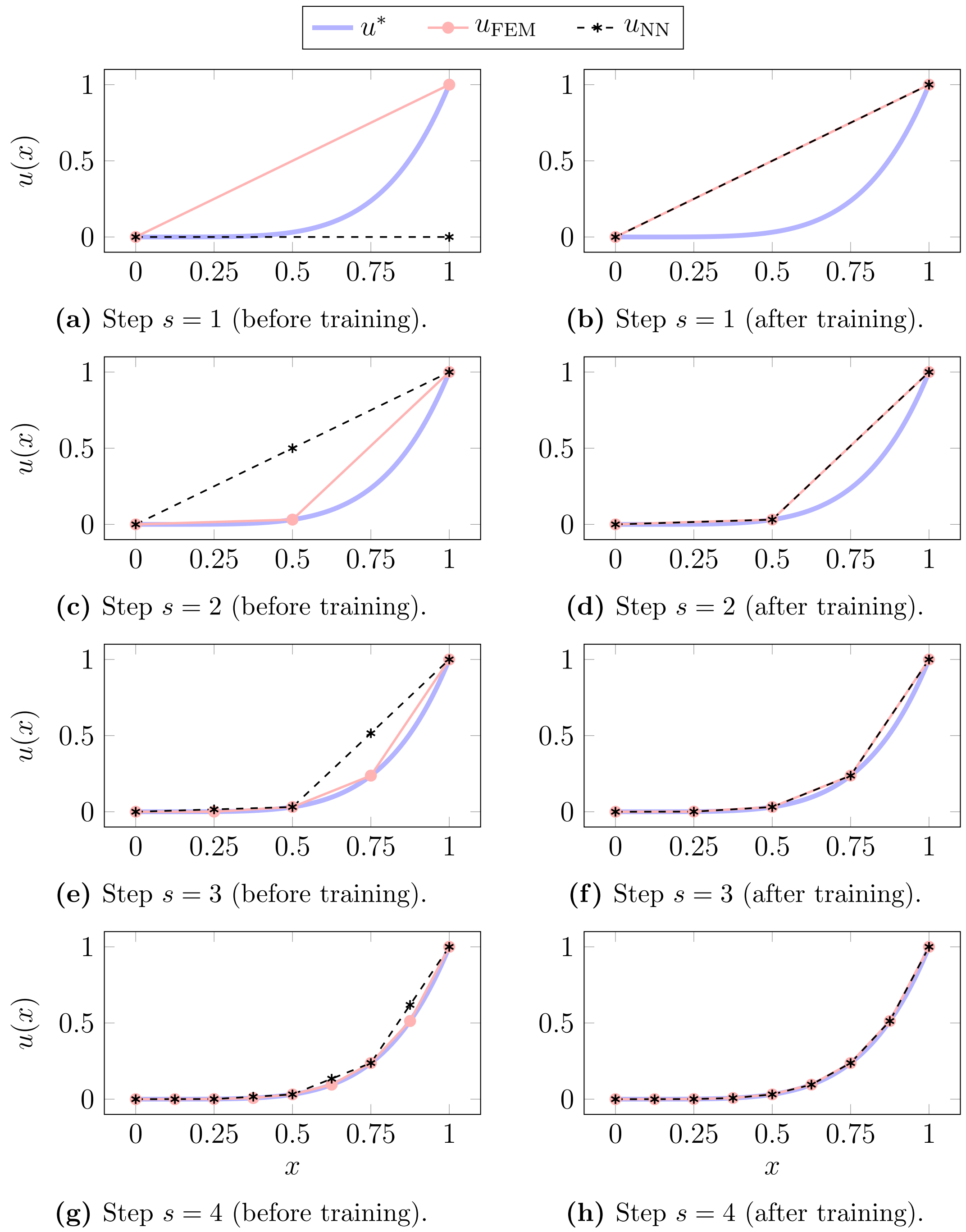
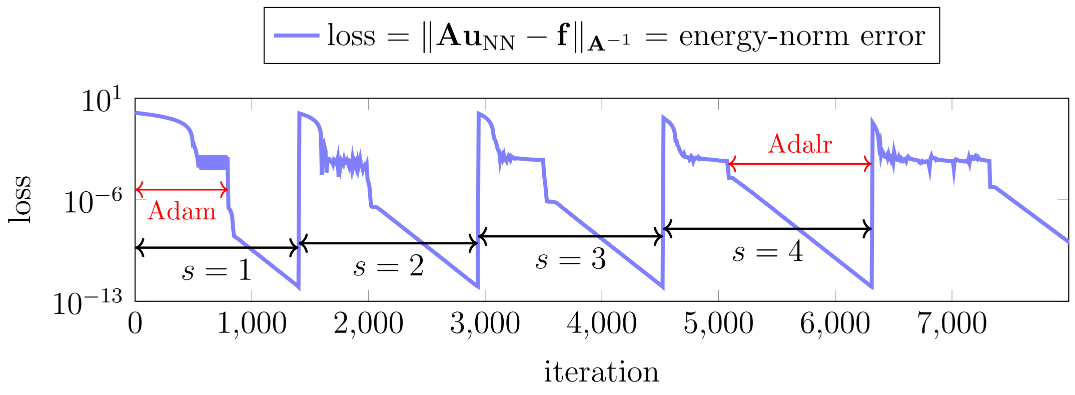
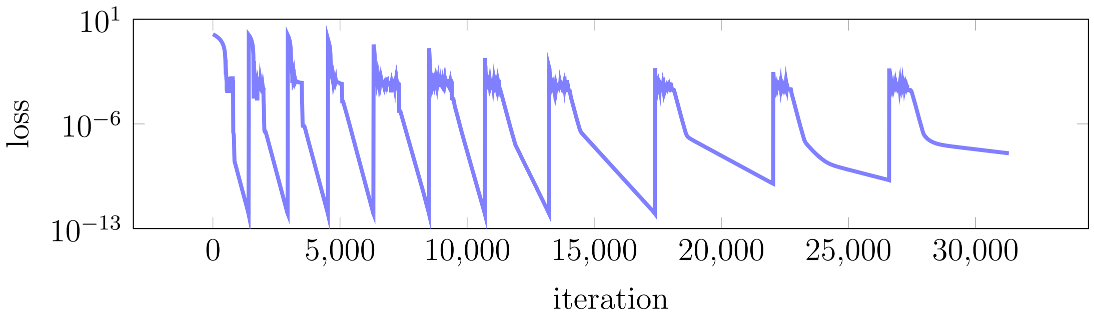


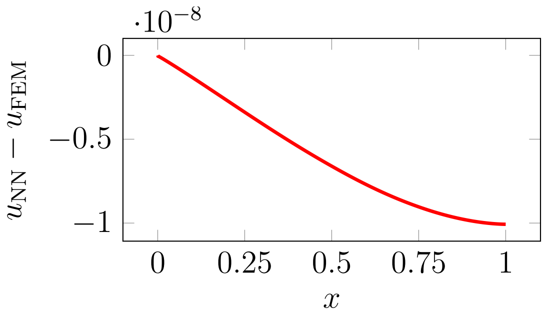
Figure B.8: First four steps of the DeepFEM for problem (B.25). u^* is the exact solution, u_{FEM} is the finite element solution, and u_{NN} is the NN prediction at each training step.



(a) Loss function evolution during the first four steps.



(b) Loss function evolution during eleven steps.



(c) Error function at the end of training.

Figure B.9: End-to-end training of DeepFEM for problem (B.25). The loss function coincides with the energy-norm error.

B.6.1.1. End-to-end vs. layer-by-layer training

By construction, each layer of DeepFEM produces the coefficients related to the basis functions associated with each Finite Element mesh. The finer the mesh, the more local the supports of the basis functions associated with the coefficients. While end-to-end training allows adjustments in the entire hierarchy of the coefficients related to basis functions of coarse and fine meshes, layer-by-layer training only adjusts coefficients associated with the finest mesh Figure B.10a shows the loss convergence under the same conditions as above but performing a layer-by-layer training. We observe that convergence deteriorates as the number of iterations increases compared to performing an end-to-end training (recall Figure B.9b). This occurs because the loss involves the gradient of the error when employing the energy-norm. Figure B.10b shows the error function, which is almost constant. Thus, the derivative of the error is nearly zero, so it is challenging to minimize the energy-norm error by adjusting individual coefficients associated with local-support basis functions since this often implies an increase in the derivative of the error. This convergence accelerates using both global-support and local-support basis functions, as in multigrid methods [37].

If we select a norm for the loss that ignores gradients, e.g., the L^2-norm, training of local-support basis functions provides outstanding results, as shown in Figure B.11a. However, optimizing with respect to the L^2-norm is discouraged for solving differential equations.

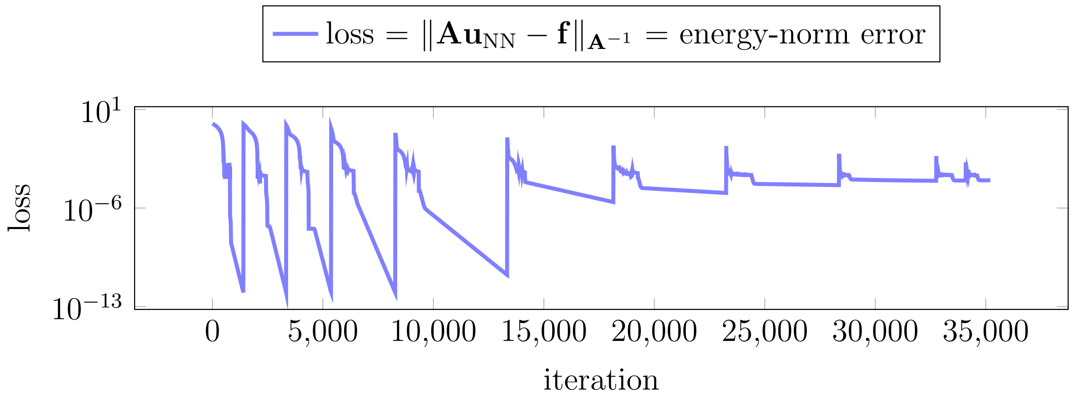
Even if the end-to-end training is the best alternative when dealing with losses involving the gradient of the residual/error, we observe a convergence deterioration of the loss function as iterations move forward (recall Figure B.9b). We suspect this is independent of the norm selection, but it occurs because of the conflicting coexistence of many trainable variables. To illustrate this, we consider the above L^2-norm case where layer-by-layer training suffices to achieve an outstanding convergence, and we perform an end-to-end training. Figure B.11b shows the convergence deterioration of the loss function.

In the following, we only consider end-to-end training cases of study.

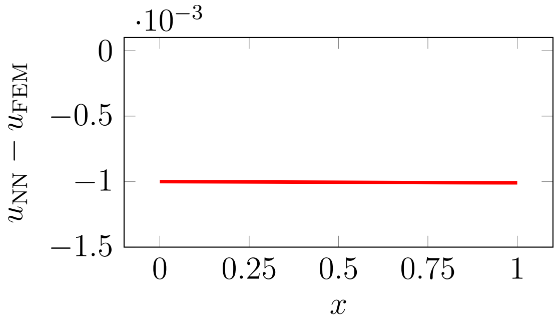
B.6.1.2. Preconditioners action

We now consider the loss given in (B.19) with three different preconditioners: (a) $\mathbf{P}$ being the identity matrix (Figure B.12a); (b) $\mathbf{P}$ being a block-Jacobi preconditioner with blocks of size two (Figure B.12b); and (c) $\mathbf{P}$ being a block-Jacobi preconditioner with blocks of size equal to half the number of elements in the mesh (Figure B.12c). In all the cases, we show the loss function evolution along with the energy-norm error evolution.

We observe some differences between the energy-norm error and the loss that

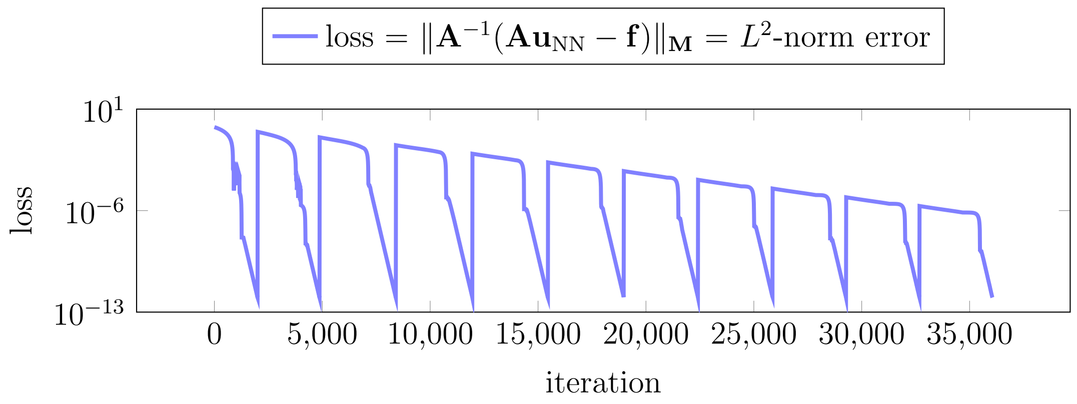


(a) Loss function evolution

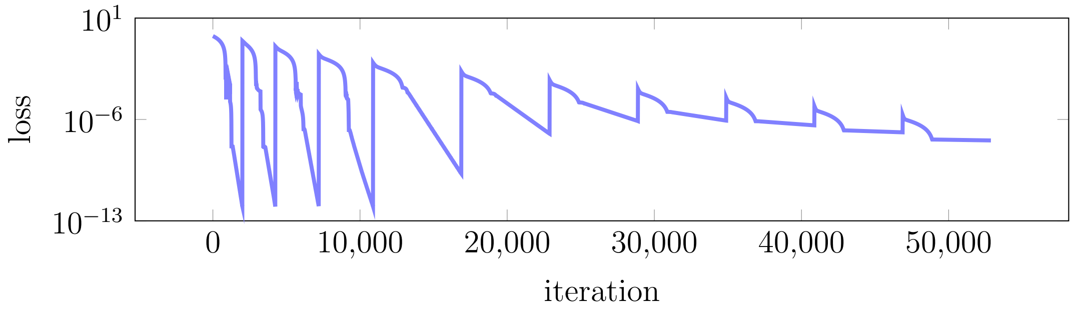


(b) Error function at the end of the training.

Figure B.10: Layer-by-layer training of the DeepFEM model for problem (B.25) employing the energy norm in the loss function.

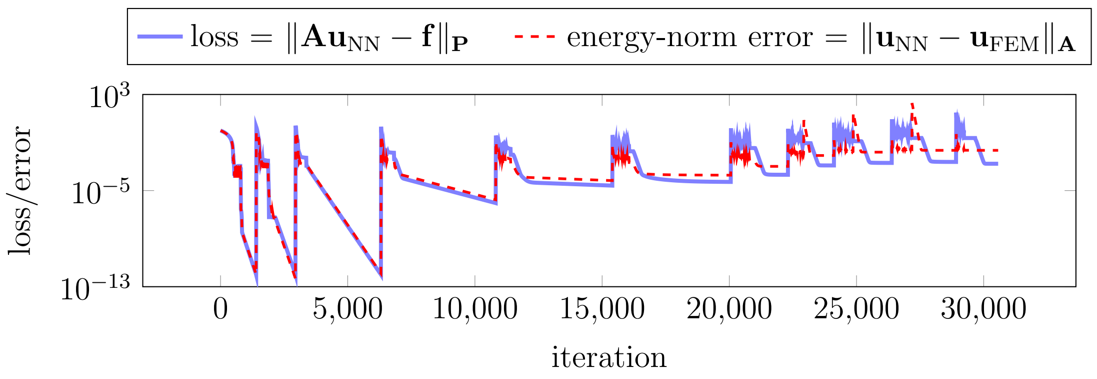


(a) Loss function evolution when perfoming a layer-by-layer training.

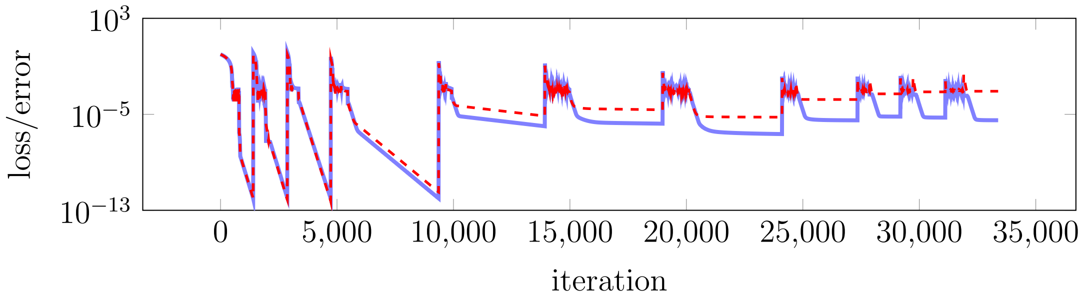


(b) Loss function evolution when performing an end-to-end training.

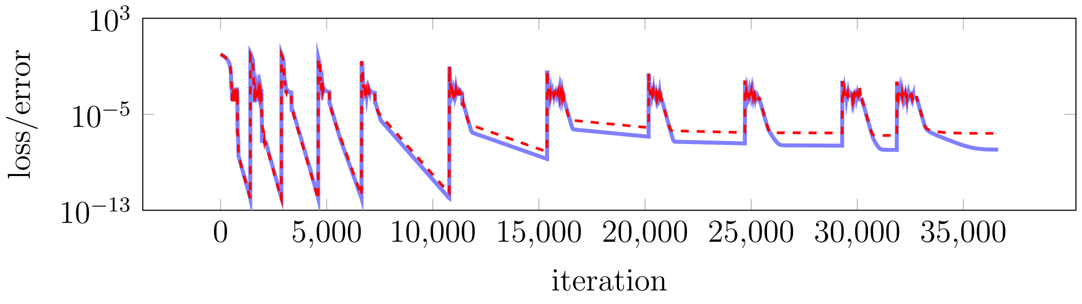
Figure B.11: Training of the DeepFEM for model problem (B.25) using the L^2-norm for the loss function.



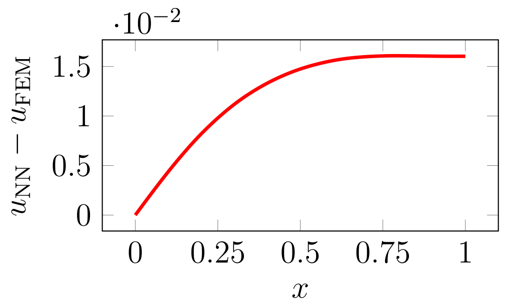
(a) Employing no preconditioner, i.e., $\mathbf{P} = \mathbf{I}$.



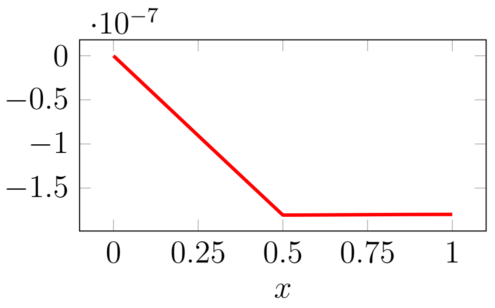
(b) Employing blocks of size two.



(c) Employing blocks of size equal to half the size of the mesh.



(d) Error function of case (a).



(e) Error function of case (c).

Figure B.12: End-to-end training of the DeepFEM for problem (B.25) using block-Jacobi preconditioners of different sizes.

increases with the mesh size, as expected. We also observe that the larger the size of the block-Jacobi, the more minor the discrepancy between the loss and the norm error. In addition, the further the loss is from the energy-norm error, the earlier the loss stagnates convergence—compare errors at Figure B.12d and Figure B.12e. This suggests that the loss induced by the energy-norm error is more convex with respect to the variables than other simplified variants, such as the loss caused by the 2-norm of the residual vector.

B.6.1.3. Norm interchange during training

Depending on the norm we select, we deal with different convexity shapes of the loss with respect to the variables of the NN. Within certain portion of the learnable variables domain, it often happens that a given loss is more convex than others, which has a direct impact on the optimizer convergence. With the aim of avoiding stagnation, we propose to change the norm during optimization, expecting to improve convexity. In this way, we consider

$$\mathcal{L}(\theta;\sigma,\alpha) = C_E\|\mathbf{A}\mathbf{u}_{\mathrm{NN}} - \mathbf{f}\|_{\mathbf{P}} + \ C_{L^2}\|\mathbf{P}(\mathbf{A}\mathbf{u}_{\mathrm{NN}} - \mathbf{f})\|_{\mathbf{M}}, \tag{B.26}$$

where $C_E, C_{L^2} \in \{0,1\}$ are distinct values that interchange when the convergence stagnates.

To illustrate the above idea, we consider the case of Figure B.12b. We maintain the loss at equation (B.19) for the first four steps. Then, we consider two variants for the loss in the fifth step when employing the Adalr optimizer: (a) same loss as in the previous steps—maintaining $C_E = 1$ in (B.26)—but with a maximum of 12,000 iterations; and (b) the loss at equation (B.26) with $C_E = 1$ for 2,000 iterations, changing to $C_{L^2} = 1$ for another 8,000 iterations, and returning to $C_E = 1$ for 2,000 additional iterations. Although the total number of iterations is the same in all situations, we obtain a loss at the end of the fifth step that is lower when performing the norm change (around 10^{-10}) than when maintaining the energy-norm (around 10^{-8})—see Figure B.13. We observe that the slope of the loss function evolution is higher when employing the L^2-norm, allowing us to start from a lower loss value when returning to the energy norm than maintaining it during the entire training step.

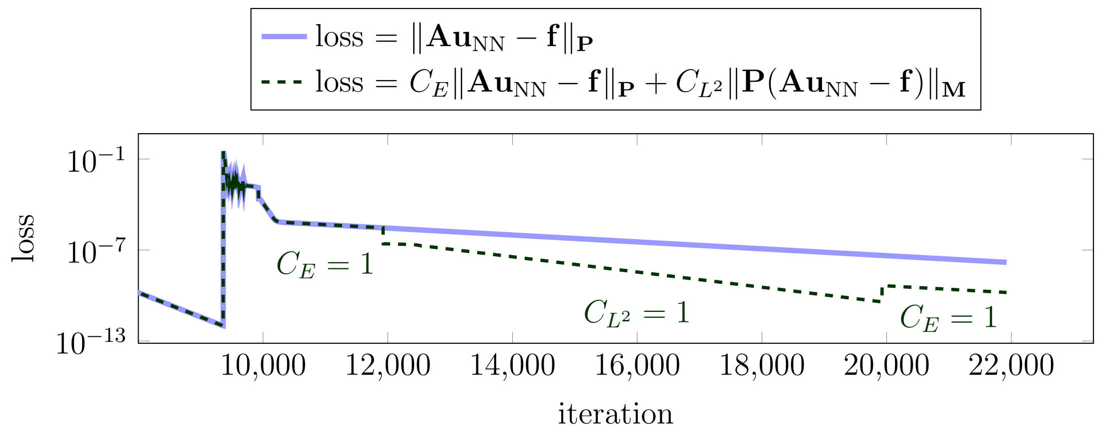


Figure B.13: Energy-norm vs. energy- and L^2-norm interchange training at step $s = 5$ in Figure B.12b.

B.6.2. Sine solution in Poisson and Helmholtz equations

We consider two different BVPs whose exact solutions are $u^*(x) = \sin(10\pi x)$:

$$\begin{cases} -u'' = 100\pi^2 \sin(10\pi x), \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \qquad \begin{cases} -u'' - 100\pi^2 u = 0, \\ u(0) = 0, u'(1) = 10\pi. \end{cases} \tag{B.27}$$

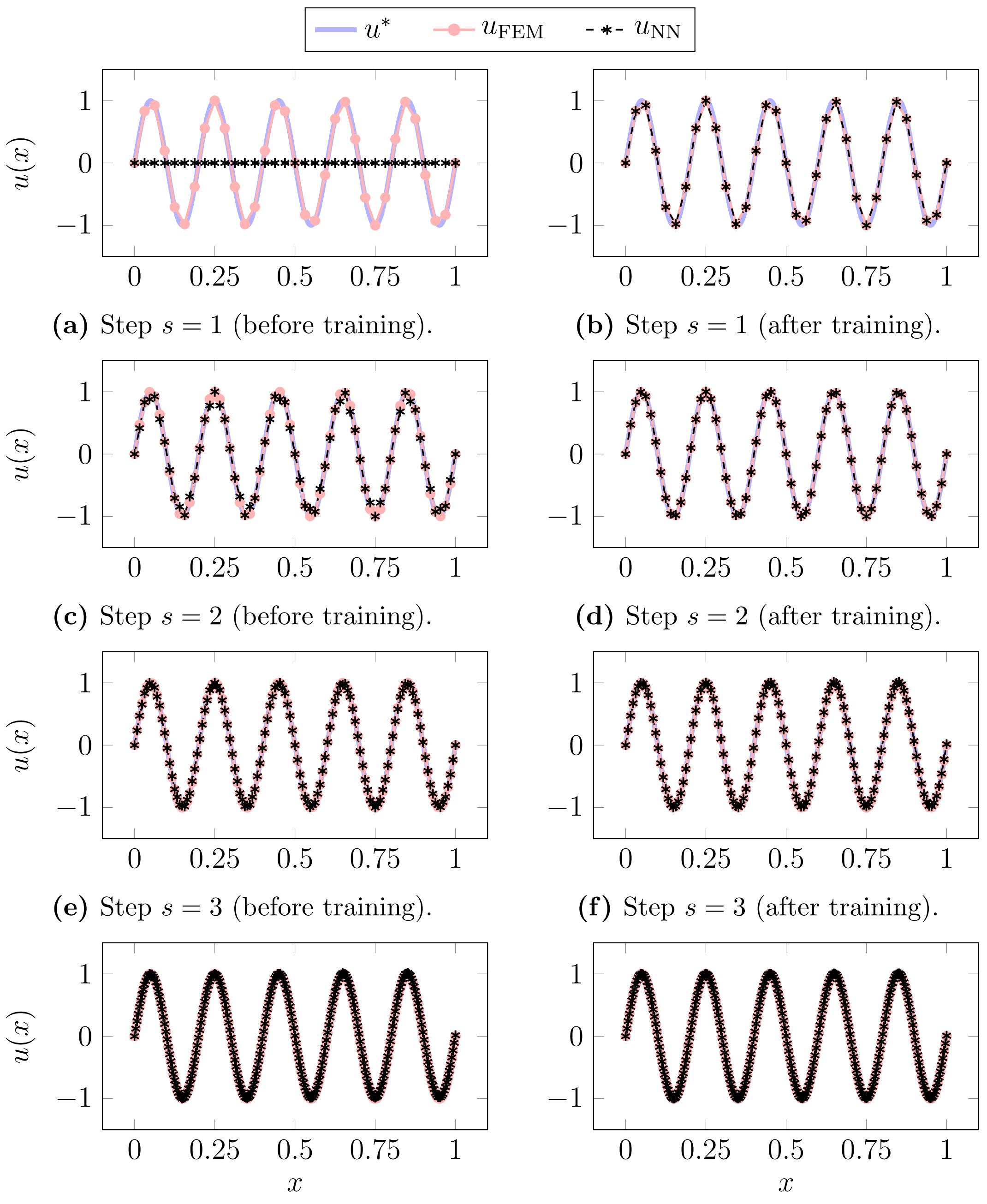
The former is a symmetric and positive-definite problem, while the second is indefinite. We solve both employing the DeepFEM selecting the loss as the H^1-norm of the preconditioned residual,

$$\mathcal{L}(\theta; \sigma, \alpha) = \|\mathbf{P}(\mathbf{A}\mathbf{u}_{\mathrm{NN}} - \mathbf{f})\|_{\mathbf{K}+\mathbf{M}}. \tag{B.28}$$

We start from 32 elements and perform three mesh refinements. We select $\mathbf{P}$ with blocks of size 32 for $1 \leq s \leq 4$. The optimizers and NN architecture are those of Section B.6.1.

Figure B.14 and Figure B.15 show the NN predictions of Poisson and Helmholtz problems, respectively. In Poisson, FEM solutions coincide with u^* at the nodal points, but this is not true for Helmholtz's equation.

When training with the preconditioner being the inverse matrix, the NN converges perfectly with a monotonic decrease. However, when considering non-inverse preconditioning, convergence stagnates after decreasing a couple of orders of magnitude (see Figure B.16a and Figure B.16b). In the Adalr optimizer training phases, the loss decreases abruptly in the early iterations but then remains flat, producing a corresponding H^1-norm error reduction that is insignificant. In both experiments, we obtain errors around 10^{-3} (see Figure B.16c and



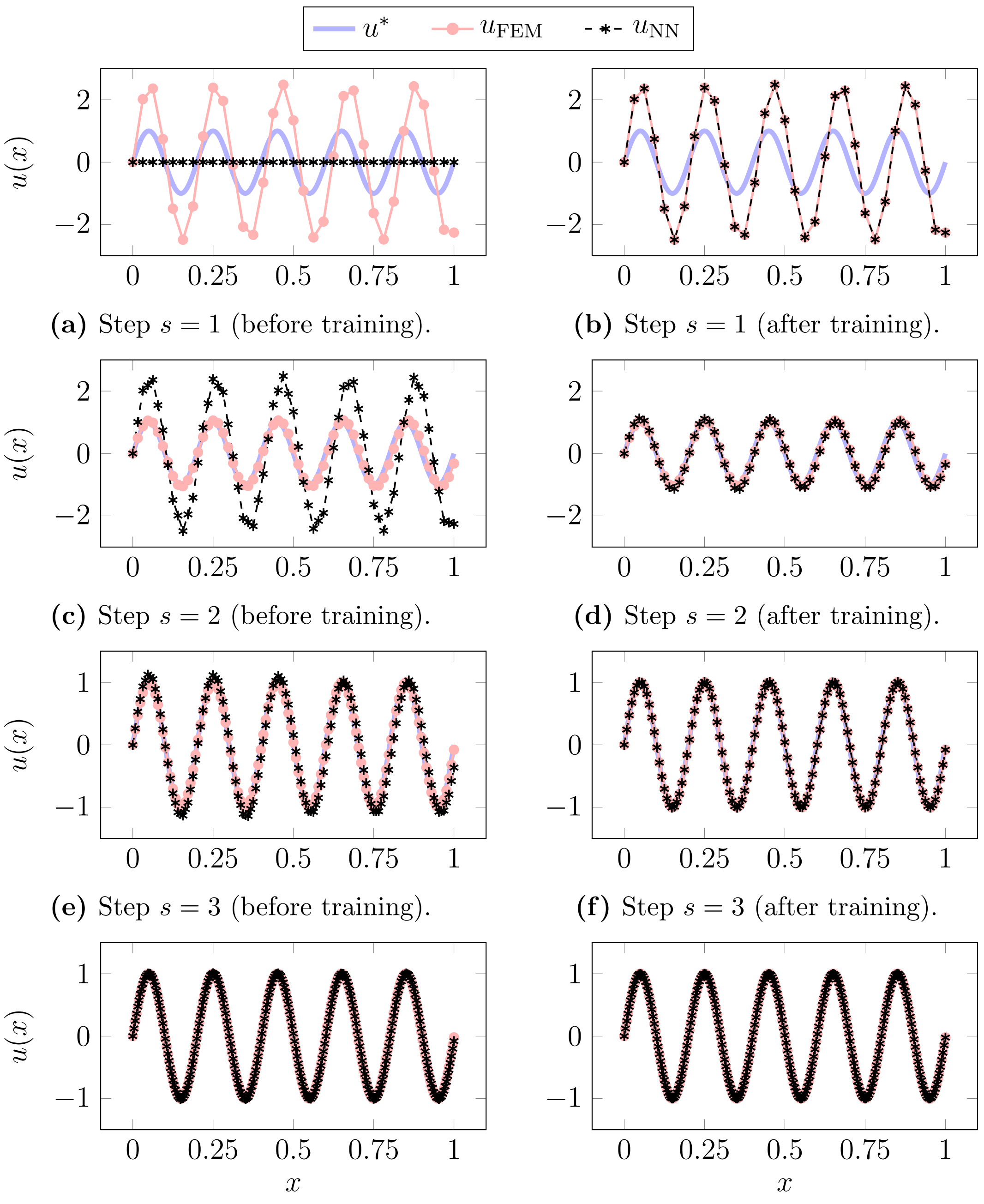
(a) Step $s = 1$ (before training). **(b)** Step $s = 1$ (after training).

(c) Step $s = 2$ (before training). **(d)** Step $s = 2$ (after training).

(e) Step $s = 3$ (before training). **(f)** Step $s = 3$ (after training).

(g) Step $s = 4$ (before training). **(h)** Step $s = 4$ (after training).

Figure B.14: Four steps for the DeepFEM in Poisson problem (B.27). u^* is the exact solution, u_{FEM} is the finite element solution, and u_{NN} is the DeepFEM prediction.



(a) Step $s = 1$ (before training).

(b) Step $s = 1$ (after training).

(c) Step $s = 2$ (before training).

(d) Step $s = 2$ (after training).

(e) Step $s = 3$ (before training).

(f) Step $s = 3$ (after training).

(g) Step $s = 4$ (before training).

(h) Step $s = 4$ (after training).

Figure B.15: Four steps of the DeepFEM in Helmholtz problem (B.27). u^* is the exact solution, u_{FEM} is the finite element solution, and u_{NN} is the DeepFEM prediction.

Figure B.16d). These results illustrate that it is possible to obtain approximate solutions with a certain (but not high) degree of accuracy, probably due to the non-convexity of the losses—recall Section 2.3.

B.6.3. Sinusoidal solution with piecewise-constant coefficients

Following Helmhotz's equation, we add varying frequencies along the propagation domain by considering piecewise-constant coefficients as follows:

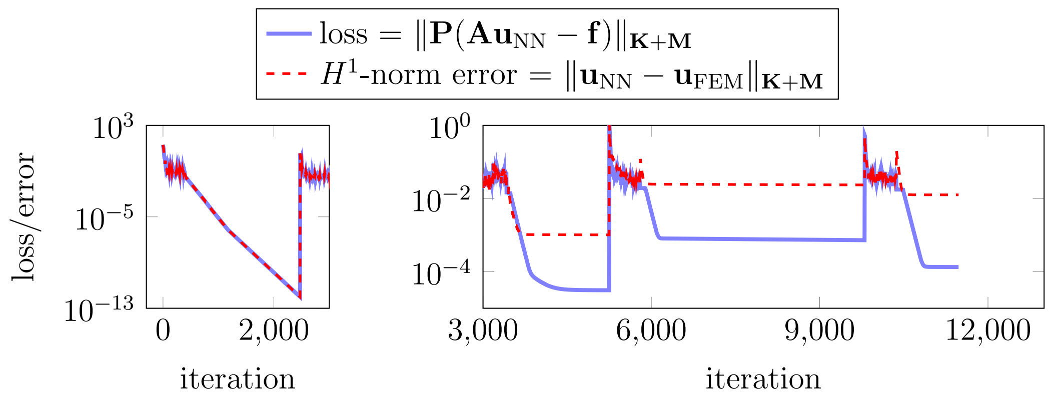
$$\begin{cases} -(\sigma u')' + \alpha u = 0, \\ u(0) = 0, u'(1) = 10\pi, \end{cases} \tag{B.29}$$

with

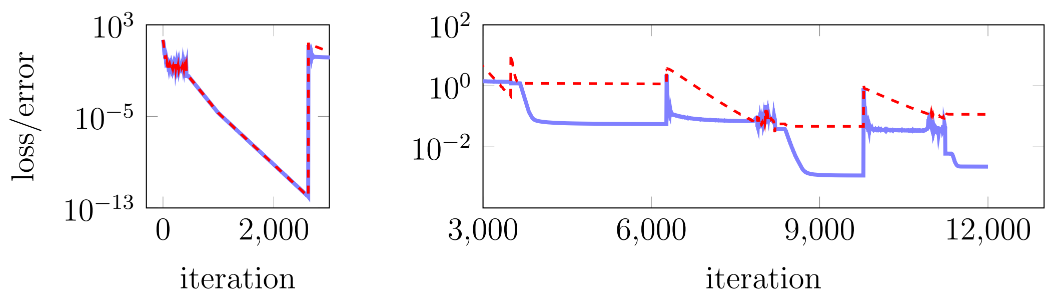
$$\sigma = \begin{cases} 1, & \text{if } 0 < x < 1/3, \\ 2, & \text{if } 1/3 < x < 2/3, \\ 3, & \text{if } 2/3 < x < 1, \end{cases} \qquad \alpha = \begin{cases} -3000, & \text{if } 0 < x < 1/3, \\ -2000, & \text{if } 1/3 < x < 2/3, \\ -1000, & \text{if } 2/3 < x < 1. \end{cases} \tag{B.30}$$

We solve it employing the H^1-norm for the loss. We start with 48 elements and perform three uniform refinements. We select $\mathbf{P}$ with 48-size blocks in the first two mesh refinements and 96-size blocks in the last two. The NN architecture is the same as the one described in Section B.6.1.

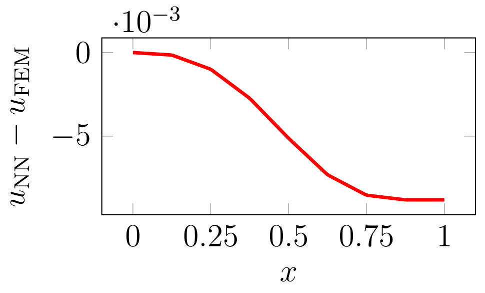
Figure B.17 shows the NN predictions and Figure B.18a shows the loss function evolution during training. The NN nicely converges when employing the inverse as a preconditioner, but in the subsequent steps, we observe a stagnation of the loss, as was previously the case. In addition, there is a decreasing influence of the loss on the H^1-norm error, which remains constant throughout the last training step, while the loss decreases by two orders of magnitude. Figure B.18b and Figure B.18c show the errors at the end of steps three and four, respectively.



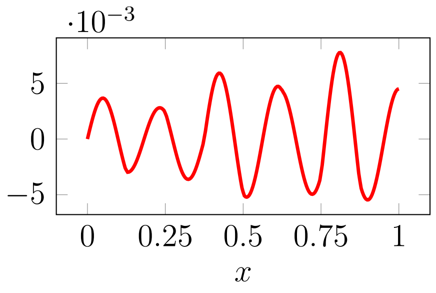
(a) Loss function evolution along with the H^1-norm error for Poisson problem (B.27).



(b) Loss function evolution along with the H^1-norm error for Helmholtz problem (B.27).

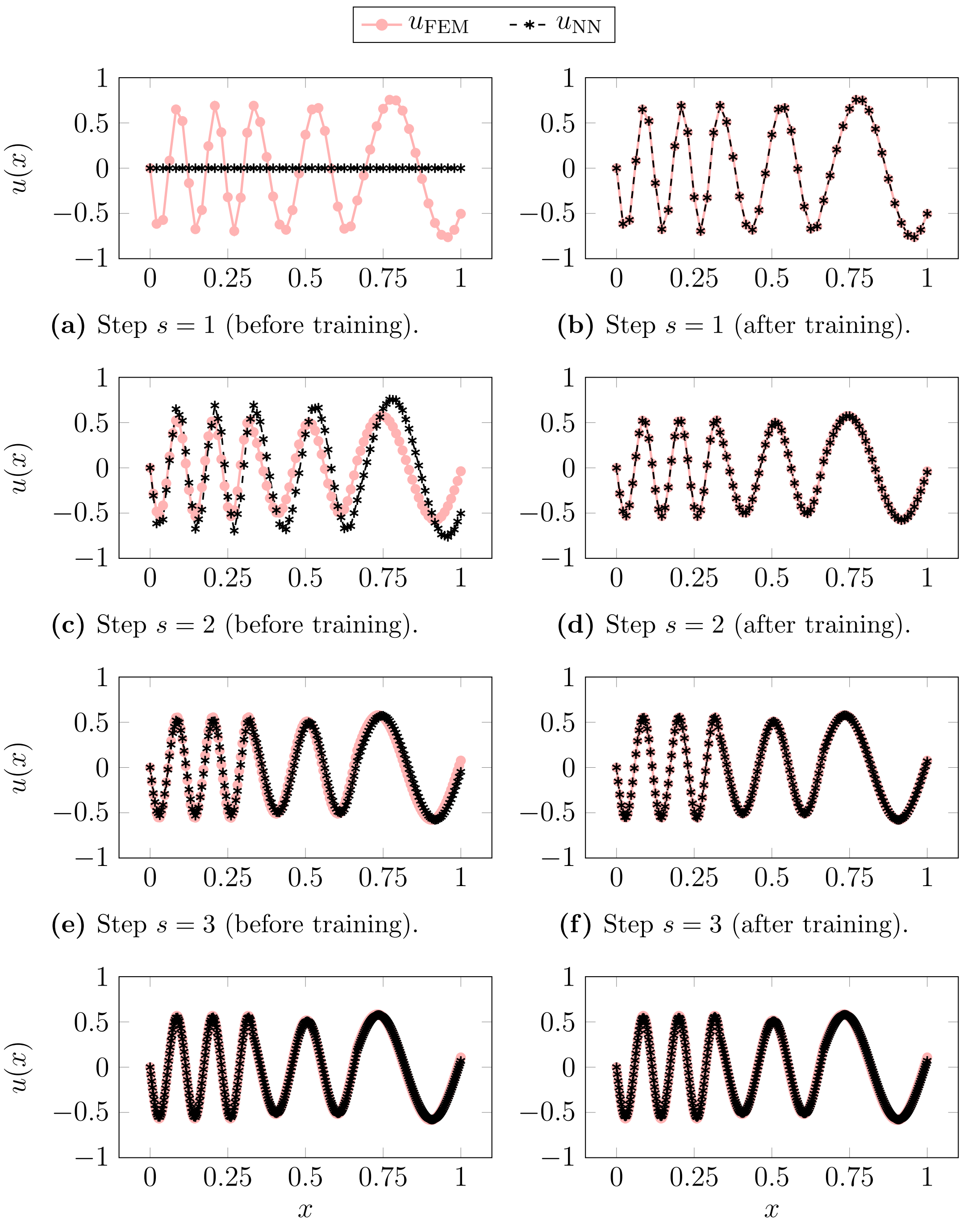


(c) Error function at the end of the training (Poisson's problem).



(d) Error function at the end of the training (Helmholtz's problem).

Figure B.16: End-to-end training of the DeepFEM for problem (B.27) along three steps with 32-size block-Jacobi preconditioners.



(a) Step $s = 1$ (before training).

(b) Step $s = 1$ (after training).

(c) Step $s = 2$ (before training).

(d) Step $s = 2$ (after training).

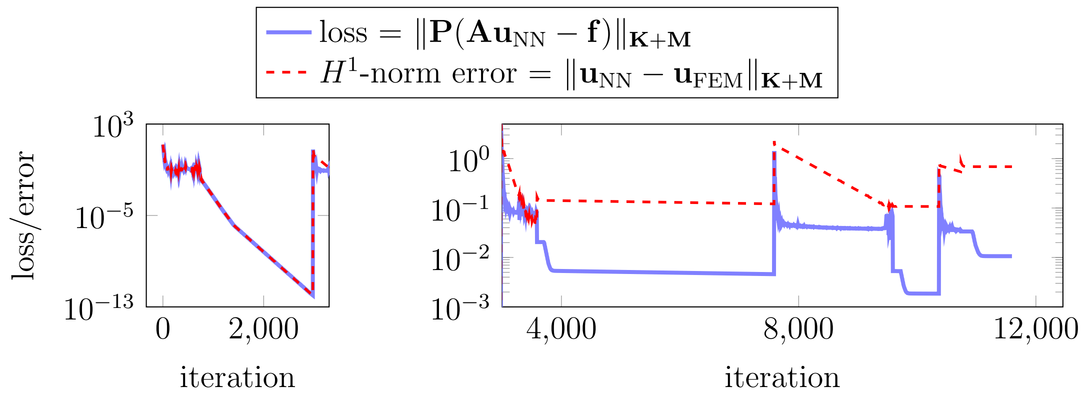
(e) Step $s = 3$ (before training).

(f) Step $s = 3$ (after training).

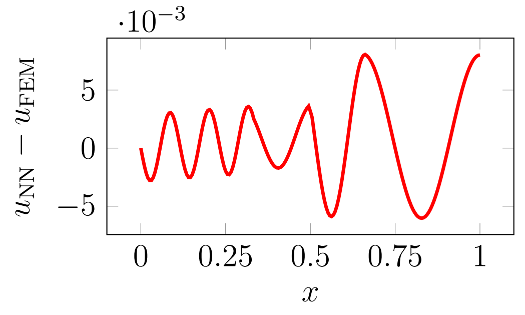
(g) Step $s = 4$ (before training).

(h) Step $s = 4$ (after training).

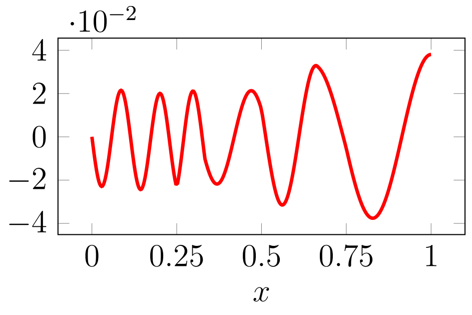
Figure B.17: Four steps of the DeepFEM for problem (B.29). u_{FEM} is the finite element solution and u_{NN} is the DeepFEM prediction.



(a) Loss function evolution along with the H^1-norm error for Helmholtz problem (B.27)



(b) Error function at the end of $s = 3$.



(c) Error function at the end of $s = 4$.

Figure B.18: End-to-end training of the DeepFEM for problem (B.29) along four steps with block-Jacobi preconditioners of sizes 48, 48, 96, and 96 at steps $s = 1, 2, 3, 4$, respectively.

B.6.4. Parametric boundary value problems

In this section, we naturally extend the DeepFEM in its parametric variant: we train the NN to learn the FEM solutions from one mesh to another for a family of PDE coefficients with fixed boundary conditions. We consider experiments varying the α coefficient with a constant parametric behavior for the following BVP:

$$\begin{cases} -u'' + \alpha u = 0, \\ u(0) = 0, u'(1) = 2\pi. \end{cases} \tag{B.31}$$

We solve it by performing three uniform refinements. For the training, we select databases of randomly selected samples of α coefficients. We train the NN on the entire database without batch partitioning. Our loss computes the norm over the preconditioned residual that depends on each sample. Specifically, we optimize with respect to an averaged sum of norms of the residual—recall Equation (B.15).

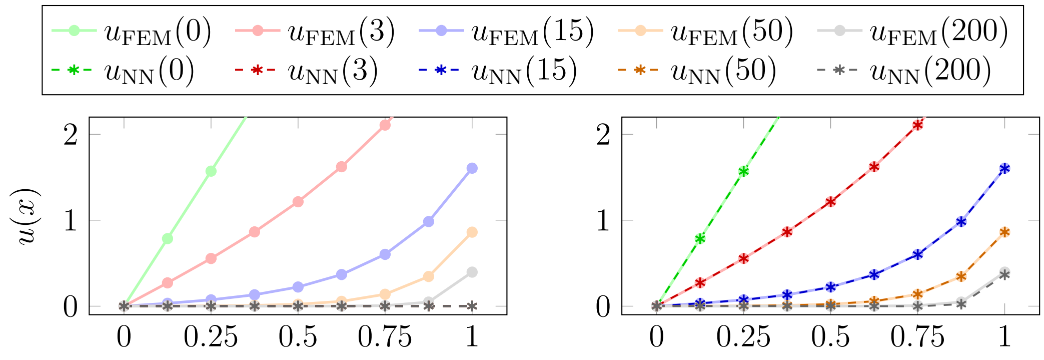
B.6.4.1. Reaction-diffusion parametric equation: $0 < \alpha < 200$

The exact solution is $u^*(x) = C(e^{\sqrt{\alpha}x} - e^{-\sqrt{\alpha}x})$ with $C = \frac{2\pi e^{\sqrt{\alpha}}}{\sqrt{\alpha}(e^{2\sqrt{\alpha}}+1)}$ and $u^*(x) \to 2\pi x$ as $\alpha \to 0$. At each step, we establish one-layer depth and 20-neuron width training block architecture for the NN. We start the DeepFEM with a uniform eight-element mesh and finish with a 64-element mesh. At each step, we employ block-Jacobi preconditioners with blocks of size eight. Since the parametric problem is symmetric and positive-definite, we consider the norm induced by the preconditioner for the optimization.

We select a database of 100 samples randomly and logarithmically distributed to train the NN. To visualize the NN performance after the training, we consider $\{0, 3, 15, 50, 200\}$ as test data, which does not take part during training. Figure B.19 shows the parametric mesh-by-mesh adaptivity of the NN over these test data. We observe a good behavior of the NN for all values of α. Table B.1 displays the losses and energy-norm errors evaluated on the test data points at the end of the training.

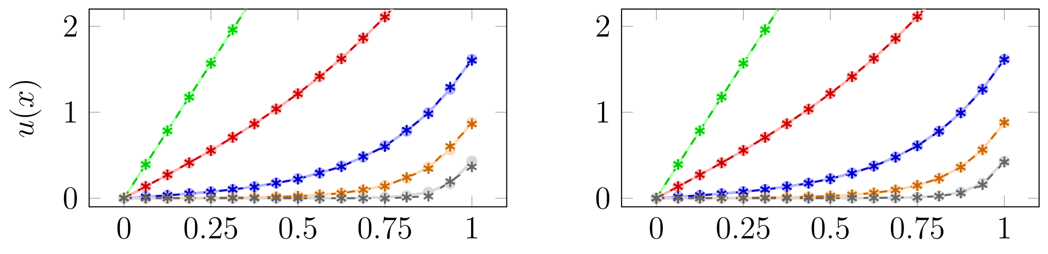
α	0	3	15	50	200
$\lVert \mathbf{A}(\alpha)\mathbf{u}_{\text{NN}}(\alpha) - \mathbf{f} \rVert_{\mathbf{P}(\alpha)}$	0.0015	0.013	0.0046	0.012	0.082
$\lVert \mathbf{u}_{\text{NN}}(\alpha) - \mathbf{u}_{\text{FEM}}(\alpha) \rVert_{\mathbf{A}(\alpha)}$	0.0160	0.014	0.0061	0.014	0.095

Table B.1: Samplewise residual and error values in the energy norm of the test data at the end of the fourth step for problem (B.31) with $0 < \alpha < 200$ when employing eight-size blocks in the preconditioners.



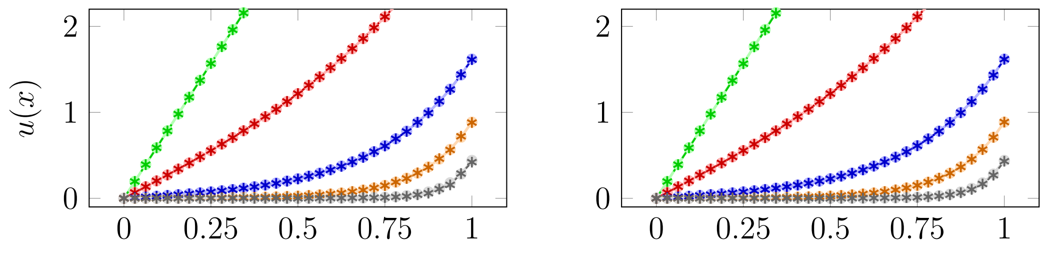
(a) Step $s = 1$ (before training).

(b) Step $s = 1$ (after training).



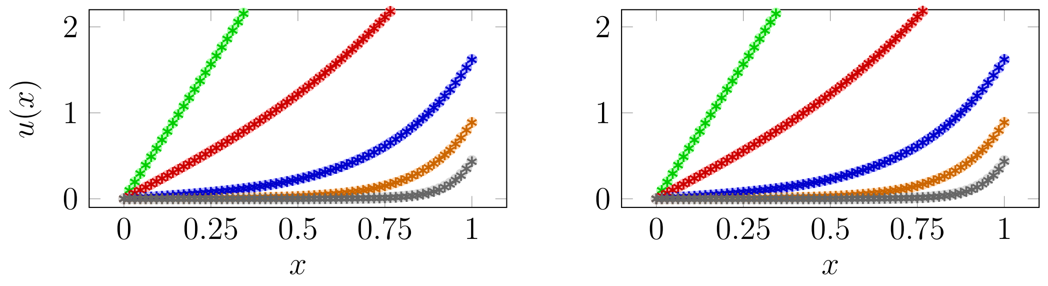
(c) Step $s = 2$ (before training).

(d) Step $s = 2$ (after training).



(e) Step $s = 3$ (before training).

(f) Step $s = 3$ (after training).

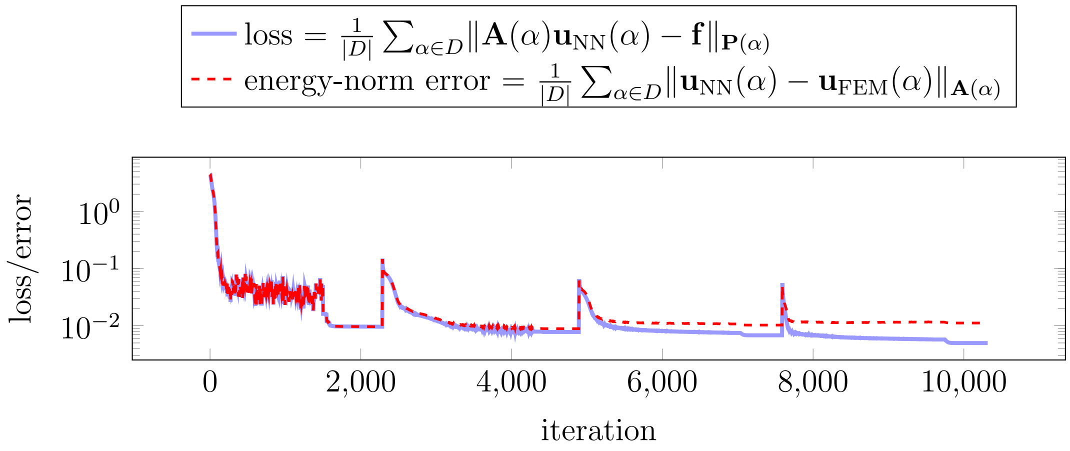


(g) Step $s = 4$ (before training).

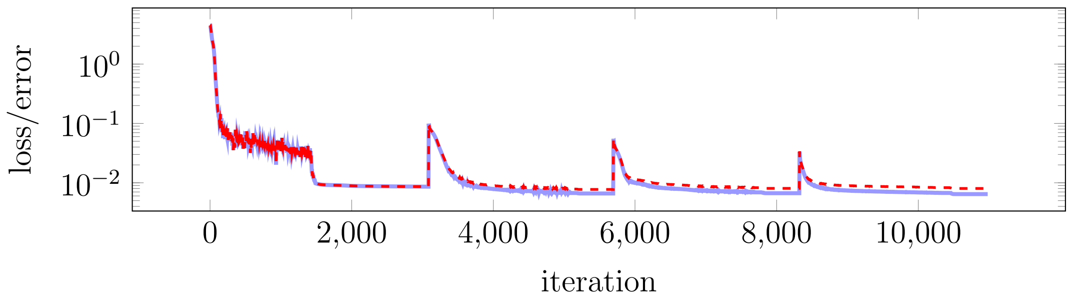
(h) Step $s = 4$ (after training).

Figure B.19: Four steps of the DeepFEM for problem (B.31) with $0 < \alpha < 200$ employing the energy norm in the loss function. $u_{\text{FEM}}(\alpha)$ and $u_{\text{NN}}(\alpha)$ denote the FEM solution and DeepFEM prediction for the α parameter coefficient, respectively.

Analyzing the evolution of the training in Figure B.20a, we observe that the loss decrease is more significant than the error in the energy norm decrease, as expected. By increasing the size of the blocks in the preconditioners step by step (with sizes 8, 8, 16, and 32 at steps one, two, three, and four, respectively), the energy-norm error does decrease in consistency with the loss (see Figure B.20b). Table B.2 displays the losses and energy-norm errors evaluated on the test data at the end of this enhanced training with preconditioning.



(a) Employing preconditioners with blocks of size eight in all steps.



(b) Employing preconditioners with blocks of sizes 8, 8, 16, and 32 at steps one, two, three, and four, respectively.

Figure B.20: Loss function evolution along with the energy-norm error during the four-step training at problem (B.31) with $0 \le \alpha \le 200$.

B.6.4.2. Helmholtz's parametric equation: $-50 < \alpha < -30$

The exact solution is $u^*(x) = C\sin(\sqrt{\alpha}x)$ with $C = \frac{2\pi}{\sqrt{\alpha}\cos(\sqrt{\alpha})}$. We consider 100 samples randomly and uniformly distributed as the training data and

$$\{-50, -45, -40, -35, -30\} \tag{B.32}$$

α	0	3	15	50	200
$\lVert\mathbf{A}(\alpha)\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{f}\rVert_{\mathbf{P}(\alpha)}$	0.0022	0.0059	0.0043	0.0086	0.011
$\lVert\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{u}_{\mathrm{FEM}}(\alpha)\rVert_{\mathbf{A}(\alpha)}$	0.0053	0.0075	0.0058	0.0101	0.013

Table B.2: Samplewise residual and error values in the energy norm of the test data at the end of the fourth step for problem (B.31) with $0 < \alpha < 200$ when employing size-increasing blocks in the preconditioners.

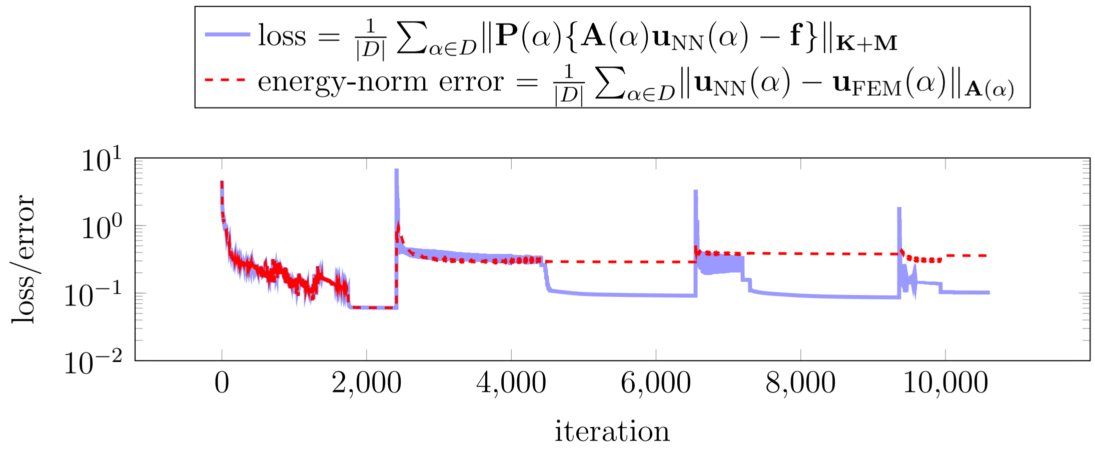
as the test data. We consider the H^1-norm for the loss and maintain the same trainable block architectures as above. If we train the NN employing the same increasing block-size criterion for the preconditioners as in Section B.6.4.2, we observe that the loss does not decrease the H^1-norm error in this occasion (Figure B.21a). If we employ the inverses as the preconditioners (Figure B.21b), the NN converges sufficiently to show adequate results. Figure B.22 shows the step-by-step predictions for the test data when the training is performed according to Figure B.21b.

Utilizing the inverses is equivalent to applying the loss:

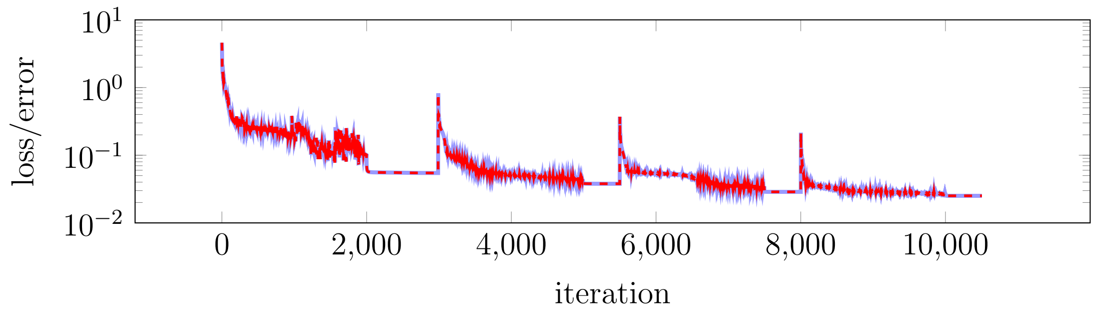
$$\mathcal{L}(\theta; D) = \frac{1}{|D|}\sum_{\alpha\in D}\lVert\mathbf{u}_{\mathrm{NN}}(\alpha)-\mathbf{u}_{\mathrm{FEM}}(\alpha)\rVert_{\mathbf{K}+\mathbf{M}}, \tag{B.33}$$

where $\mathbf{u}_{\mathrm{NN}}(\alpha)$ stands for the DeepFEM prediction and $\mathbf{u}_{\mathrm{FEM}}(\alpha)$ for the FEM solution vectors for the α parameter. Precalculating all these vectors and employing (B.33) as the loss during training could be significantly less expensive than computing/dealing with the inverse matrices. Along this work, we have computed inverses to be consistent with the presented arguing, and we refrained from reporting times since an efficient solver should never invert a matrix explicitly [5, 169, 158].

Go to Chapter 4 for the English reading continuation/Joan 4. kapitulura ingelesezko irakurketarekin jarraitzeko.

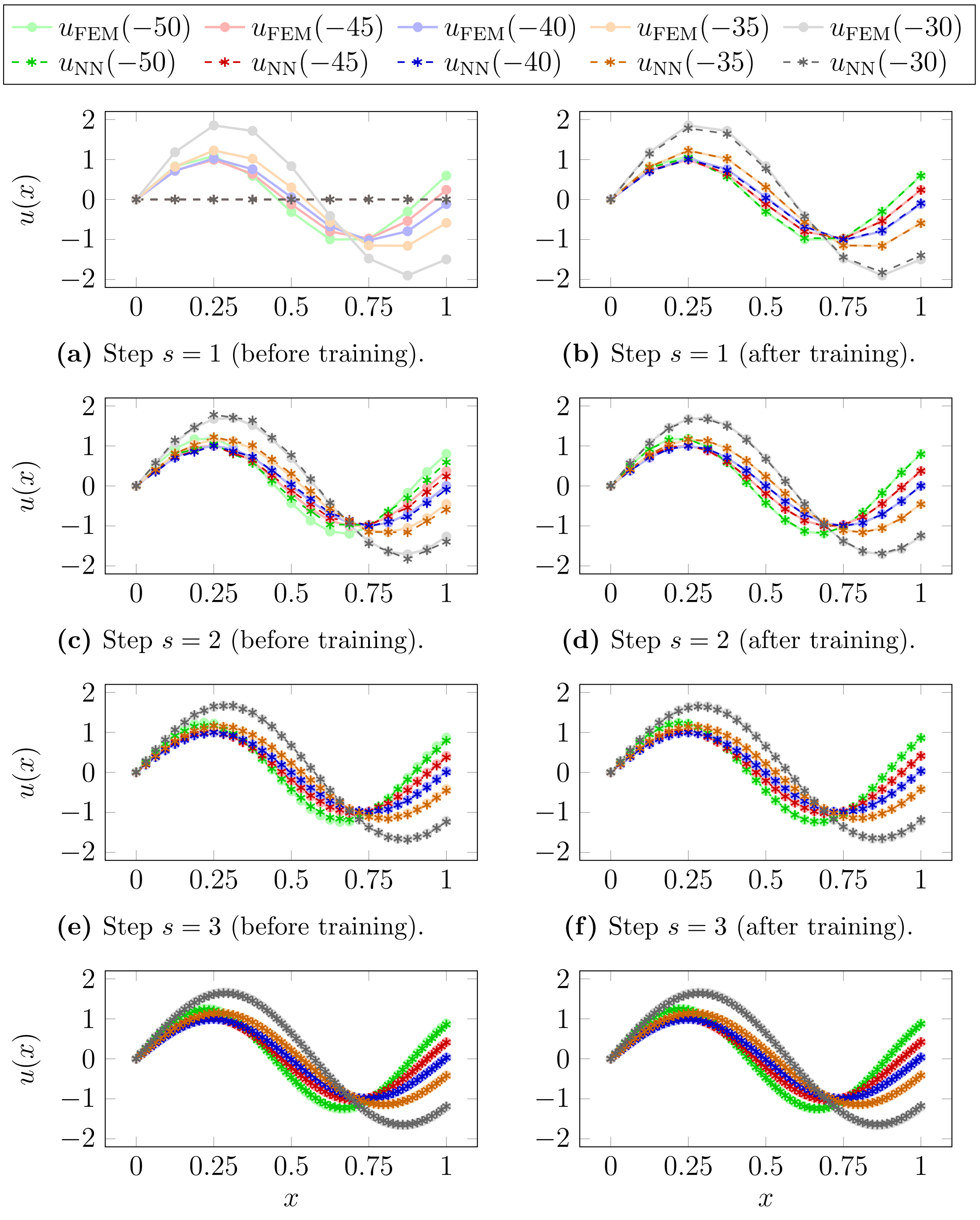


(a) Employing block-Jacobi preconditioners of sizes equal to 8, 8, 16 and 32 at the first, second, third, and fourth steps, respectively.



(b) Employing inverses as preconditioners.

Figure B.21: Loss function evolution along H^1-norm error during the four-step training at problem (B.31) with $-50 \leq \alpha \leq -30$.



(a) Step $s = 1$ (before training).

(b) Step $s = 1$ (after training).

(c) Step $s = 2$ (before training).

(d) Step $s = 2$ (after training).

(e) Step $s = 3$ (before training).

(f) Step $s = 3$ (after training).

(g) Step $s = 4$ (before training).

(h) Step $s = 4$ (after training).

Figure B.22: Four steps of the DeepFEM for problem (B.31) with $-50 < \alpha < -30$ employing the H^1-norm error in the loss function. $u_{\mathrm{FEM}}(\alpha)$ and $u_{\mathrm{NN}}(\alpha)$ denote the FEM solution and DeepFEM prediction for the α parameter coefficient, respectively.

C. Memory-based Monte Carlo integration (Chapter 5)

Summary. Monte Carlo integration is a widely used quadrature rule to solve Partial Differential Equations using Neural Networks due to its ability to guarantee overfitting-free solutions and high-dimensional scalability. However, this stochastic method produces noisy losses and gradients during training, which hinders a proper convergence diagnosis. Typically, this is overcome using an immense (disproportionate) amount of integration points, which deteriorates the training performance. This work proposes a memory-based Monte Carlo integration method that produces accurate integral approximations without requiring the high computational costs of processing large samples during training. *Refer to [219] for the published version.*

C.1. Introduction

Using a deterministic quadrature rule with fixed integration points possibly leads to misbehavior of the NN away from the integration points (an overfitting issue) [186], producing significant integration errors and, consequently, poor solutions. To overcome this, *Monte Carlo (MC) integration* is a popular and suitable choice of quadrature rule due to the mesh-free and stochastic sampling of the integration points during training [128, 84, 62, 48]. However, MC integration error is of order $\mathcal{O}(1/\sqrt{N})$, where N is the number of integration points [163]. Thus, in practice, this may require tens or hundreds of thousands of integration points to obtain an acceptable error per integral approximation—even for one-dimensional problems, which deteriorates the training speed.

In this work, we propose a memory-based approach that approximates definite integrals involving NNs by taking advantage of the information gained in previous iterations. As long as the expected value of these integrals does not change significantly, this technique reduces the expected integration error and leads to better approximations. Moreover, since gradients are also described in terms of definite integrals, we apply this approach to the gradient computations, which reinterprets the well-known momentum method [177] when we appropriately modify the hyperparameters of the optimizer.

The remainder of this chapter is organized as follows. Section C.2 reviews the approximation and optimization frameworks on NNs, Section C.3 proposes the memory-based integration and optimization strategies, and Section C.4 relates these proposals with the momentum method.

C.2. Review of the approximation, discretization, parameterization, and optimization setups

We summarize many of the concepts already discussed in Chapter 2 to (re)define notation and present the method in question more fluently for the reader.

Let us consider a well-defined minimization problem of the form

$$u^* = \arg\min_{u\in\mathbb{U}} \mathcal{F}(u), \tag{C.1}$$

where $\mathbb{U}$ denotes the search space of functions with domain Ω, $\mathcal{F} : \mathbb{U} \longrightarrow \mathbb{R}$ is the objective function governing our minimization problem, and u^* is the exact solution.

Let $u_\theta : \Omega \longrightarrow \mathbb{R}$ denote a neural network architecture parameterized by the set of trainable parameters θ with domain Θ. Then, a NN approximation of problem (C.1) consists in replacing the continuum-level search space $\mathbb{U}$ with the parameterized search space $\mathbb{U}_\Theta = \{u_\theta : \theta \in \Theta\}$.

To carry out the minimization, we resort to a first-order gradient-descent scheme that is described as follows:

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t}), \tag{C.2}$$

where $\lambda > 0$ is the learning rate and θ_t denotes the trainable parameters at the t^{th} iteration.

For $\mathcal{F}$ in the form of a definite integral,

$$\mathcal{F}(v_\theta) = \int_\Omega I(v_\theta)(x)\ dx, \tag{C.3}$$

we approximate it by a quadrature rule, producing a *loss function* $\mathcal{L}$. Considering MC integration as the quadrature rule for the loss, we have

$$\mathcal{F}(v_\theta) \approx \mathcal{L}(v_\theta) := \frac{\text{Vol}(\Omega)}{N} \sum_{i=1}^{N} I(v_\theta)(x_i), \tag{C.4}$$

where $\{x_i\}_{i=1}^N$ is a stochastic set of integration points sampled from a random uniform distribution in Ω.

Similarly, for the gradients, we have

$$\frac{\partial \mathcal{F}}{\partial \theta}(v_\theta) \approx g(v_\theta) := \frac{\partial \mathcal{L}}{\partial \theta}(v_\theta) = \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} \frac{\partial I(v_\theta)}{\partial \theta}(x_i), \tag{C.5}$$

which yields a discretized version of (C.2), so-known as the SGD optimizer [187],

$$\theta_{t+1} := \theta_t - \lambda g(v_{\theta_t}). \tag{C.6}$$

From now on, we write $\mathcal{F}(\theta_t)$, $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$, $\mathcal{L}(\theta_t)$, and $g(\theta_t)$ as simplified versions of $\mathcal{F}(v_{\theta_t})$, $\frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t})$, $\mathcal{L}(v_{\theta_t})$, and $g(v_{\theta_t})$, respectively[1].

C.3. Memory-based integration and optimization

If we train the network according to (C.6), we obtain a noisy and oscillatory behavior of the loss and the gradient. This occurs because of the introduced MC integration error at each training iteration. Figure C.1 (blue curve) illustrates the noisy behavior of MC integration in a network with a single trainable parameter that permits exact calculation of $\mathcal{F}$ (black curve).

This section is divided into three parts. Section C.3.1 introduces the considered model problem for experimentation, Section C.3.2 describes the proposed memory-based MC integration rule, and Section C.3.3 extends such memory-based scheme to the gradients for training.

C.3.1. Model problem

Let us consider the model problem $-u'' = 4\delta_{1/2}$ in $\Omega = (0,1)$ with homogeneous Dirichlet conditions on $\partial\Omega = \{0,1\}$. Then, its weak form reformulation possesses the following bilinear and linear forms:

$$b(u,v) = \int_0^1 u'v', \qquad l(v) = v(1/2), \qquad u \in \mathbb{U} = H_0^1(0,1) = \mathbb{V} \ni v. \tag{C.7}$$

Its unique exact solution is $u^*(x) = 2x$ in $0 \leq x \leq 1/2$ and $u^*(x) = 2(1-x)$ in $1/2 \leq x \leq 1$. Moreover, it admits a Ritz-type minimization reformulation as follows:

$$u^* = \arg\min_{u \in \mathbb{U}} \frac{1}{2} \int_0^1 [u'(x)]^2 dx - u(1/2). \tag{C.8}$$

[1]Note that we have dropped the *realization map* terminology—recall (2.5)—when expressing the dependence of the objective function on the parameters. That is, $\mathcal{F}(\theta)$ directly refers to $(\mathcal{F} \circ \Phi_{\mathrm{NN}})(\theta)$ according to the notation developed in Chapter 2.

Noticing that $2\tanh(\theta(x - 1/2))$ has arbitrary approximation capacity for $(u^*)'$—recall Section 2.4, integrating and imposing the boundary conditions, we obtain the (atypical) NN architecture with arbitrary approximation capacity for u^* given by

$$u_\theta(x) = \frac{2}{\theta}\Big\{ \log\big[\cosh(\theta/2)\big] - \log\big[\cosh(\theta(1/2 - x))\big]\Big\}. \tag{C.9}$$

Consequently, straightforward calculations yield:

$$\mathcal{F}(\theta) = 2 - \frac{4}{\theta}\tanh(\theta/2) - \frac{8}{\theta}\log\big[\cosh(\theta/2)\big], \tag{C.10a}$$

$$\frac{\partial\mathcal{F}}{\partial\theta}(\theta) = \frac{-4(\theta-1)\tanh(\theta/2) - 2\theta\operatorname{sech}^2(\theta/2) + 8\log[\cosh(\theta/2)]}{\theta^2}, \tag{C.10b}$$

for $\theta \neq 0$. Notice that approximation is achieved as $\theta \to \infty$ with corresponding $\mathcal{F}(\theta) \to -2$ and $\frac{\partial\mathcal{F}}{\partial\theta}(\theta) \to 0$.

C.3.2. Integration

In order to decrease the integration error during training, we replace (C.4) with the following recurrence process:

$$\mathcal{F}(\theta_t) \approx \mathcal{L}_t := \begin{cases} \mathcal{L}(\theta_0), & t = 0, \\ \alpha_t\mathcal{L}(\theta_t) + (1-\alpha_t)\mathcal{L}_{t-1}, & t \geq 1, \end{cases} \tag{C.11a}$$

where $\mathcal{L}(\theta_t)$ is the MC estimate at the t^{th} iteration, and $\{\alpha_t\}_{t\geq 0}$ is a selected sequence of coefficients $0 < \alpha_t \leq 1$ such that $\alpha_0 = 1$. In expanded form,

$$\mathcal{L}_t = \sum_{l=0}^{t} \alpha_l \left(\prod_{s=1}^{t-l} (1 - \alpha_{l+s}) \right) \mathcal{L}(\theta_l), \tag{C.11b}$$

which shows that the approximation $\mathcal{L}_t$ of $\mathcal{F}(\theta_t)$ is indeed a linear combination of the current and all previous MC integration estimates. If $\alpha_t = 1$ for all $t \geq 0$, we recover the usual MC integration case without memory.

Figure C.1 (red curve) shows the memory-based loss $\mathcal{L}_t$ evolution along training according to ordinary SGD optimization (C.6) and selecting $\alpha_t = e^{-0.001t} + 0.001$. Initially, we integrate with large errors (α_t is practically one, and therefore, there is hardly any memory in $\mathcal{L}_t$). However, as we progress in training, we increasingly endow memory to $\mathcal{L}_t$ (α_t becomes small), and as a consequence, its integration error decreases. $\mathcal{L}_t$ produces more accurate approximations than $\mathcal{L}(\theta_t)$, allowing better convergence monitoring to, for example, establish proper stopping criteria during training.

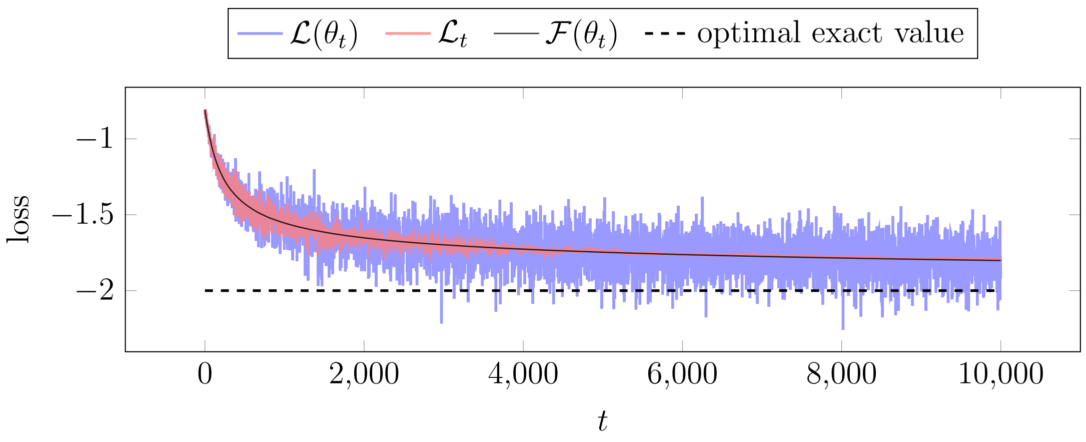


Figure C.1: Training of the single-trainable-parameter NN presented in Section C.3.1 whose architecture permits exact calculation of $\mathcal{F}$. The training is performed according to (C.6), and thus, $\mathcal{L}(\theta_t)$ is its associated loss function. $\mathcal{L}_t$ and $\mathcal{F}(\theta_t)$ are computed for monitoring.

C.3.3. Optimization

While the proposed schemeimproves the approximation of $\mathcal{F}$, we have that a corresponding SGD scheme using (C.11) is equivalent to classical SGD with a learning rate that is multiplied by α_t.

We can naturally endow the idea of memory-based integration to the gradients, as $g(\theta_t)$ is also obtained via MC integration—recall (C.5),

$$\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t) \approx g_t := \begin{cases} g(\theta_0), & t = 0, \\ \gamma_t g(\theta_t) + (1-\gamma_t) g_{t-1}, & t \geq 1, \end{cases} \tag{C.12a}$$

where $\{\gamma_t\}_{t\geq 0}$ is a selected sequence of coefficients such that $0 < \gamma_t \leq 1$ and $\gamma_0 = 1$. Then, we obtain a memory-based SGD optimizer employing the g_t term instead of $g(\theta_t)$—recall (C.6),

$$\theta_{t+1} := \theta_t - \lambda g_t. \tag{C.12b}$$

In the expanded form, we have

$$g_t = \sum_{l=0}^{t} \gamma_l \left(\prod_{s=1}^{t-l} (1 - \gamma_{l+s}) \right) g(\theta_l). \tag{C.12c}$$

Figure C.2 shows the gradient evolution during training of the previous single-trainable-parameter model problem. We select $\alpha_t = \gamma_t$ for all t, with the same

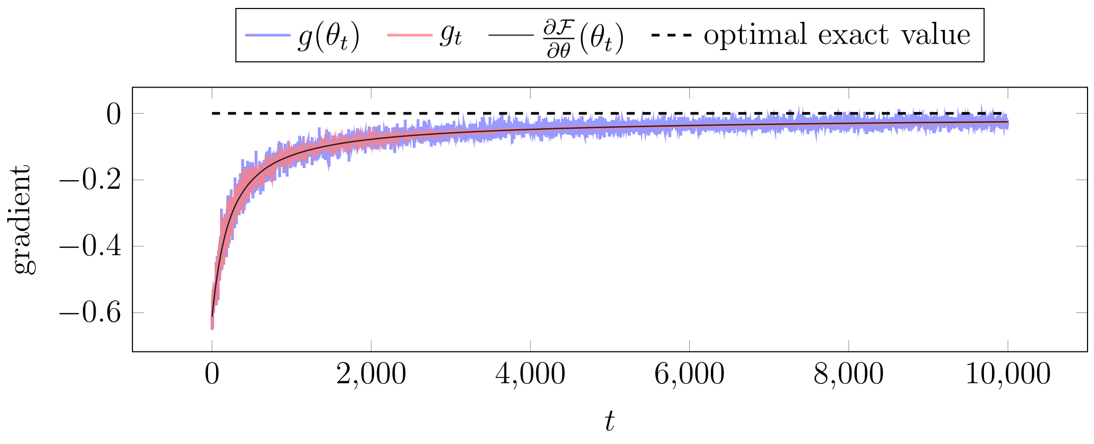


Figure C.2: Gradient evolution of the single-parameter model problem in Figure C.1 during training. The optimization is performed according to (C.6) using $g(\theta_t)$, while g_t and $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$ are computed for monitoring.

exponential decay as before. g_t produces more accurate approximations of the exact gradients than $g(\theta_t)$, minimizing the noise.

Proper tuning of the coefficients α_t and γ_t is critical to maximize integration performance. Coefficients should be high (low memory) when the involved integrals vary rapidly (e.g., at the beginning of training). Conversely, when the approximated solution is near equilibrium and the relevant integrals vary slowly, the coefficients should be low (high memory).

C.4. Relation with the momentum method

The SGD optimizer with momentum (SGDM) [177, 209] is commonly introduced as the following two-step recursive method:

$$v_{t+1} := \beta v_t - g(\theta_t), \tag{C.13a}$$

$$\theta_{t+1} := \theta_t + \lambda v_{t+1}, \tag{C.13b}$$

where v_t is the *momentum accumulator* initialized by $v_0 = 0$, and $0 \leq \beta < 1$ is the momentum coefficient. If $\beta = 0$, we recover the classical SGD optimizer (C.6). Rewriting (C.13) in terms of the scheme $\theta_{t+1} = \theta_t - \lambda g_t$, we obtain

$$g_t = g(\theta_t) + \beta g_{t-1} = \sum_{l=0}^{t} \beta^{t-l} g(\theta_l). \tag{C.13c}$$

A more sophisticated version of the SGDM modifies the momentum coefficient during training (see, e.g., [50]), namely, defined as in (C.13) but replacing β with $\beta_t \in [0,1)$ for some conveniently selected sequence $\{\beta_t\}_{t\geq 1}$. Then, the g_t term results

$$g_t = g(\theta_t) + \beta_t g_{t-1} = \sum_{l=0}^{t} \left(\prod_{s=1}^{t-l} \beta_{l+s} \right) g(\theta_l). \tag{C.14}$$

Selecting proper hyper-parameters β_t during training is challenging. However, by readjusting the learning rate and momentum coefficient in the SGDM optimizer according to γ_t in (C.12) for $t \geq 1$,

$$\lambda_t := \lambda_{t-1} \frac{\gamma_t}{\gamma_{t-1}}, \qquad \lambda_0 := \lambda, \tag{C.15a}$$

$$\beta_t := \gamma_{t-1} \frac{1-\gamma_t}{\gamma_t}, \tag{C.15b}$$

we recover our memory-based proposal (C.12).

Both optimizations (C.12) and (C.13)–(C.14) stochastically accumulate gradients to readjust the trainable parameters. However, while (C.13)–(C.14) considers a geometrically weighted average of past gradients without aiming g_t resemble $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$, our proposal (C.12) re-scales the current and prior gradients so g_t intends to imitate $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$—recall Figure C.2.

(C.15) reinterprets the SGDM as an exact-gradient performer by re-scaling the learning rate. In contrast, our memory-based proposal provides the exact-gradient interpretation, leaving the learning rate free. Consequently, the learning rate is an independent hyperparameter of the gradient-based optimizer and not an auxiliary element to interpret gradients during training. Moreover, our optimizer is designed to work in parallel with the memory-based loss (C.11) that approximates the (typically unavailable) objective function during training.

Go to Chapter 6 for the English reading continuation/Joan 6. kapitulura ingelesezko irakurketarekin jarraitzeko.

D. Achievements (Chapter 7)

This dissertation contributes to the solution of Partial Differential Equations (PDEs) using Artificial Neural Networks (ANNs), providing three main novel contributions as a result of the successful accomplishment of a series of activities developed throughout the PhD program.

Below, we collect the most relevant research contributions and stays carried out since enrolling in the Mathematics and Statistics PhD program of the University of the Basque Country (UPV/EHU) in October 2019.

D.1. Publications

Peer-reviewed published works.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-Based Monte Carlo Integration for Solving Partial Differential Equations Using Neural Networks.* In: Mikyška, J., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds) Computational Science – ICCS 2023. Lecture Notes in Computer Science, Vol. 14074, 2023. Springer, Cham.
https://doi.org/10.1007/978-3-031-36021-3_51.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method (D^2RM) for solving Partial Differential Equations using Neural Networks.* Computer Methods in Applied Mechanics and Engineering (Q1, Top 1%), Vol. 405, 115892, 2023.
https://doi.org/10.1016/j.cma.2023.115892.

2022 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Computer Methods in Applied Mechanics and Engineering (Q1, Top 1%), Vol. 391, 114562, 2022.
https://doi.org/10.1016/j.cma.2021.114562.

D.2. Conferences

Contributions to national and international conferences. Underlined appears the speaker of the presentation.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *The Deep Double Ritz Method: a Deep Learning Residual Minimization Method for solving Partial Differential Equations.* 2nd IACM Mechanistic Machine Learning and Digital Engineering for Computational Science and Technology Conference. University of Texas at El Paso, USA. September 24-27, 2023.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-based Monte Carlo integration for solving Partial Differential Equations using Neural Networks.* XXXI COMCA – Congreso de Matemática Capricornio, Antofagasta, Chile. August 2-4, 2023.

2023 Carlos Uriarte, Jamie M. Taylor, David Pardo, Oscar A. Rodríguez, Patrick Vega. *Memory-based Monte Carlo integration for solving Partial Differential Equations using Neural Networks.*. ICCS 2023 – 23rd International Conference on Computational Science. Prague, Czech Republic. July 3-5, 2023.

2023 Carlos Uriarte, David Pardo, Ignacio Muga. *Goal-Oriented Deep Ritz and Least-Squares methods.* ICCS 2023 – 23rd International Conference on Computational Science. Prague, Czech Republic. July 3-5, 2023.

2022 Carlos Uriarte, David Pardo, and Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* SIAM MDS22 – Conference on Mathematics of Data Science. San Diego, California, USA. September 26-30, 2023.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *Solving Partial Differential Equations using Adversarial Neural Networks.* CMN 2022 – Congress on Numerical Methods in Engineering. Las Palmas de Gran Canaria, Spain. September 12-14,2022.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *Adversarial Neural Networks for solving variationally formulated Partial Differential Equations.* WCCM/APCOM 2022 – 15th World Congress on Computational Mechanics and 8th Asian Pacific Congress on Computational Mechanics. Yokohama, Japan. July 31 – August 5, 2022.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *An Adversarial Networks approach for solving Partial Differential Equations.* ICCS 2022 – 22nd International Conference on Computational Science. London, UK. June 21-23, 2022.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, and Judit Muñoz-Matute. *A Generative Adversarial Networks approach for solving Partial Differential Equations.* ECCOMAS 2022 – 8th European Congress on Computational Methods in Applied Sciences and Engineering. Oslo, Norway. June 5-9, 2022.

2021 Carlos Uriarte, Ángel J. Omella, David Pardo. *A Finite Element based Deep Learning solver for parametric PDEs.* ICCS 2021 – 21st International Conference on Computational Science. Krakow, Poland. June 16-18, 2021.

D.3. Courses, Seminars & Workshops

Contributions to courses, seminars, and workshops. Underlined appears the main contributor. Lack of underlining means that there is no specific principal contributor.

2023 Carlos Uriarte, David Pardo. *(Goal-Oriented) Deep Residual Minimization Methods.* Seminar at the Faculty of Computer Science, Electronics, and Telecommunications of the AGH University of Science and Technology. Krakow, Poland. July 11, 2023.

2023 Tomás Teijeiro, Ángel J. Omella, Jamie M. Taylor, Carlos Uriarte, David Pardo. *Parametric PDEs using Deep Learning.* Organizer of the Course and Working Group. Asturias, Spain. May 21-27, 2023.

2023 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations using Neural Networks.* Oral presentation at the Workshop Numerical Methods in Geophysics: Present, Future, and Applications held in Valparaíso, Chile. January 12-13, 2023.

2023 Ángel J. Omella, Carlos Uriarte, and David Pardo. *Coding Deep Neural Networks for PDEs.* Professor at the Course organized by the Pontificia Universidad Católica de Valparaíso and held in Olmué, Chile. January 15-20, 2023.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations.* Oral presentation at the XC Encuentro Anual de la Sociedad Matemática de Chile in Punta de Tralca, Chile. December 8-10, 2022.

2022 Carlos Uriarte, David Pardo, Ignacio Muga, Judit Muñoz-Matute. *A Deep Double Ritz Method for solving Partial Differential Equations.* Seminar at the Pontificia Universidad Católica de Valparaíso, Chile. November 11, 2022.

2022 David Pardo, Magdalena Strugaru, Jamie M. Taylor, Ángel J. Omella, Jon A. Rivera, Carlos Uriarte, Ignacio Muga, Judit Muñoz-Matute. *Deep Learning for Simulation and Inversion Problems.* Plenary talk at the Oden Institute for Computational Engineering and Sciences of the University of Texas at Austin, Texas, USA. October 21, 2022.

2022 David Pardo, Ángel J. Omella, Jamie M. Taylor, Carlos Uriarte, Jon A. Rivera, Magdalena Strugaru. *Deep Learning for Simulation and Inversion Problems.* Plenary talk at the 9º Congreso Metropolitano de Modelado y Simulación Numérica held in Mexico. May 4-6, 2022.

2021 David Pardo, Ángel J. Omella, Jon Ander River, Carlos Uriarte, Ana Fernádez-Navamuel. *Solving Forward and Inverse Problems with Deep Learning.* Organizer of the Course and Working Group held in Cantabria, Spain. May 10-18, 2021.

2021 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Seminar at the Pontificia Universidad Católica de Valparaíso, Chile. March 26, 2021.

2021 Carlos Uriarte, David Pardo, Ángel J. Omella. *A Finite Element based Deep Learning solver for parametric PDEs.* Lecture at the Faculty of Computer Science, Electronics, and Telecommunications of the AGH University of Science and Technology in Krakow, Poland. January 21, 2021.

2020 David Pardo, Ángel J. Omella, Carlos Uriarte. *Deep Learning for Solving Inverse Problems using TF2.0.* Organizer of the Course and Working Group held in Asturias, Spain. June 28 – July 4, 2020.

D.4. Research Stays

Research stays abroad longer than 30 days.

2023 Instituto de Matemáticas de la Pontificia Universidad Católica de Vaparaíso, Chile. Supervisor: Prof. Ignacio Muga (92 days).

2022 Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, USA. Supervisor: Prof. Leszek F. Demkowicz (37 days).

D.5. Disseminating Activities

Disseminating talks.

2023 Elisabete Alberdi, Carlos Uriarte. *Matematika eguneroko bizitzan. Ekuazio diferentzialak mundua azaltzeko.* Disseminating talk at Uhagon Kulturgunea in Markina-Xemein, Bizkaia, Spain. May 12, 2023.

2022 Carlos Uriarte. *Ekuazio diferentzialak mundua azaltzeko eta sare-neuronal artifizialak horiek ebazteko.* Disseminating talk at the 19th Edition of the Mathematics in Everyday Life (Matemáticas en la vida cotidiana) lecture series at Bidebarrieta Kulturgunea, Bilbao, Spain. May 19, 2022.

Bibliography/Bibliografia

[1] L. P. Aarts and P. Van Der Veer. Neural network method for solving partial differential equations. *Neural Processing Letters*, 14:261–271, 2001.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[4] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), 2018.

[5] A. Abur. A parallel scheme for the forward/backward substitutions in solving sparse linear equations. *IEEE transactions on power systems*, 3(4):1471–1478, 1988.

[6] M. Ainsworth and J. T. Oden. A posteriori error estimation in finite element analysis. *Computer methods in applied mechanics and engineering*, 142(1-2):1–88, 1997.

[7] A. Al-Aradi, A. Correia, G. Jardim, D. de Freitas Naiff, and Y. Saporito. Extensions of the deep Galerkin method. *Applied Mathematics and Computation*, 430:127287, 2022.

[8] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Archive of numerical software*, 3(100), 2015.

[9] J. Alvarez-Aramberri. *hp-Adaptive Simulation and Inversion of Magnetotelluric Measurements.* PhD thesis, University of the Basque Country (UPV/EHU) and University of Pau (UPPA), 2015.

[10] J. Alvarez-Aramberri, D. Pardo, and H. Barucq. Inversion of Magnetotelluric Measurements Using Multigoal Oriented hp-adaptivity. *Procedia Computer Science*, 18:1564–1573, 2013.

[11] S. Alyaev, M. Shahriari, D. Pardo, Á. J. Omella, D. S. Larsen, N. Jahani, and E. Suter. Modeling extra-deep electromagnetic logs using a deep neural network. *GEOPHYSICS*, 86(3):E269–E281, 2021.

[12] S. Amari. A Theory of Adaptive Pattern Classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, 1967.

[13] A. Aminataei and M. M. Mazarei. Numerical solution of Poissons equation using radial basis function networks on the polar coordinate. *Computers & Mathematics with Applications*, 56(11):2887–2895, 2008.

[14] J. D. Anderson and J. Wendt. *Computational Fluid Dynamics*, volume 206. Springer, 1995.

[15] O. Arndt, T. Barth, B. Freisleben, and M. Grauer. Approximating a finite element model by neural network prediction for facility optimization in groundwater engineering. *European journal of operational research*, 166(3):769–781, 2005.

[16] R. C. Aster, B. Borchers, and C. H. Thurber. *Parameter Estimation and Inverse Problems.* Elsevier, 2019.

[17] I. Babuška. Error-bounds for finite element method. *Numerische Mathematik*, 16(4):322–333, 1971.

[18] I. Babuvška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM Journal on Numerical Analysis*, 15(4):736–754, 1978.

[19] F. Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

[20] G. Bao, X. Ye, Y. Zang, and H. Zhou. Numerical solution of inverse problems by weak adversarial networks. *Inverse Problems*, 36(11):115003, 2020.

[21] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[22] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[23] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[24] B. Bauer and M. Kohler. On Deep Learning as a remedy for the curse of dimensionality in nonparamtric regression. *The Annals of Statistics*, 47(4):2261–2285, 2019.

[25] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical computer science*, 22(3):317–330, 1983.

[26] M. Baxter and A. Rennie. *Financial Calculus: An Introduction to Derivative Pricing*. Cambridge University Press, 1996.

[27] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[28] R. S. Beidokhti and A. Malek. Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. *Journal of the Franklin Institute*, 346(9):898–913, 2009.

[29] R. E. Bellman. Dynamic programming. *New Jersey Google Scholar*, pages 24–73, 1957.

[30] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[31] A. I. Beltzer and T. Sato. Neural classification of finite elements. *Computers & structures*, 81(24-25):2331–2335, 2003.

[32] M. S. Berger. *Nonlinearity and functional analysis: lectures on nonlinear problems in mathematical analysis*, volume 74. Academic press, 1977.

[33] M. Bertero and P. Boccacci. *Introduction to Inverse Problems in Imaging.* CRC Press, 2020.

[34] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[35] J. Blechschmidt and O. G. Ernst. Three ways to solve partial differential equations with neural networksA review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

[36] J. P. Boyd. *Chebyshev and Fourier spectral methods.* Courier Corporation, 2001.

[37] J. H. Bramble. *Multigrid methods.* Chapman and Hall/CRC, jan 2019.

[38] F. Brauer and C. Castillo-Chavez. *Mathematical Models in Population Biology and Epidemiology.* Texts in Applied Mathematics. Springer New York, 2011.

[39] S. C. Brenner. *The mathematical theory of finite element methods.* Springer, 2008.

[40] I. Brevis, I. Muga, and K. G. van der Zee. Neural control of discrete weak formulations: Galerkin, least squares & minimal-residual methods with quasi-optimal weights. *Computer Methods in Applied Mechanics and Engineering*, 402:115716, 2022.

[41] J. Brunken, K. Smetana, and K. Urban. (Parametrized) First Order Transport Equations: Realization of Optimally Stable Petrov–Galerkin Methods. *SIAM Journal on Scientific Computing*, 41(1):A592–A621, 2019.

[42] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

[43] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801, 2021.

[44] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *Journal of Computational Physics*, 443:110514, 2021.

[45] Z. Cai, J. Chen, M. Liu, and X. Liu. Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic PDEs. *Journal of Computational Physics*, 420:109707, 2020.

[46] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral methods: evolution to complex geometries and applications to fluid dynamics.* Springer Science & Business Media, 2007.

[47] H. Chen, L. Kong, and W.-J. Leng. Numerical solution of PDEs via integrated radial basis function networks with adaptive training algorithm. *Applied Soft Computing*, 11(1):855–860, 2011.

[48] J. Chen, R. Du, P. Li, and L. Lyu. Quasi-Monte Carlo sampling for solving partial differential equations by deep neural networks. *Numerical Mathematics. Theory, Methods and Applications*, 14(2):377–404, 2021.

[49] J. Chen, R. Du, and K. Wu. A comparison study of deep Galerkin method and deep Ritz method for elliptic problems with different boundary conditions. *arXiv preprint arXiv:2005.04554*, 2020.

[50] J. Chen, C. Wolfe, Z. Li, and A. Kyrillidis. Demon: Improved neural network training with momentum decay. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3958–3962. IEEE, 2022.

[51] G. Claeskens, N. L. Hjort, et al. *Model selection and model averaging*, volume 330. Cambridge University Press Cambridge, 2008.

[52] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[53] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and whats next. *Journal of Scientific Computing*, 92(3):88, 2022.

[54] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[55] L. Demkowicz and J. Gopalakrishnan. A class of discontinuous Petrov–Galerkin methods. II. Optimal test functions. *Numerical Methods for Partial Differential Equations*, 27(1):70–105, 2011.

[56] L. Demkowicz, J. Gopalakrishnan, and B. Keith. The DPG-star method. *Computers & Mathematics with Applications*, 79(11):3092–3116, 2020.

[57] L. Demkowicz, P. Monk, L. Vardapetyan, and W. Rachowicz. deRham Diagram for *hp* Finite Element Spaces. *Computers & Mathematics with Applications*, 39(7-8):29–38, 2000.

[58] L. F. Demkowicz and J. Gopalakrishnan. An overview of the discontinuous Petrov Galerkin method. *Recent Developments in Discontinuous Galerkin Finite Element Methods for Partial Differential Equations: 2012 John H Barrett Memorial Lectures*, pages 149–180, 2014.

[59] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[60] J. Deng, Z. Yue, L. Tham, and H. Zhu. Pillar design by combining finite element methods, neural networks and reliability: a case study of the Feng Huangshan copper mine, China. *International Journal of Rock Mechanics and Mining Sciences*, 40(4):585–599, 2003.

[61] J. E. Dennis, Jr and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[62] J. Dick, F. Y. Kuo, and I. H. Sloan. High-dimensional integration: the quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

[63] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving Partial Differential Equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[64] T. Dozat. Incorporating Nesterov Momentum into Adam, 2016.

[65] C. Duan, Y. Jiao, Y. Lai, D. Li, J. Z. Yang, et al. Convergence Rate Analysis for Deep Ritz Method. *Communications in Computational Physics*, 31(4):1020–1048, 2022.

[66] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[67] S. Dupond. A thorough review on the current advance of neural network structures. *Annual Reviews in Control*, 14(14):200–230, 2019.

[68] W. E and B. Yu. The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, 2017.

[69] W. E and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[70] S. J. Farlow. *Partial Differential Equations for Scientists and Engineers.* Courier Corporation, 1993.

[71] A. A. Fernandez. The application of feedforward artificial neural networks to function approximation and the solution of differential equations. Master's thesis, Rice University, 1994.

[72] R. P. Feynman. *The Feynman Lectures on Physics Vol l.* Narosa, 1986.

[73] N. A. Gershenfeld. *The nature of mathematical modeling.* Cambridge University Press, 1999.

[74] V. Girault and P.-A. Raviart. *Finite element methods for Navier-Stokes equations: theory and algorithms*, volume 5. Springer Science & Business Media, 2012.

[75] A. Golbabai, M. Mammadov, and S. Seifollahi. Solving a system of nonlinear integral equations by an RBF network. *Computers & Mathematics with Applications*, 57(10):1651–1658, 2009.

[76] A. Golbabai and S. Seifollahi. Radial basis function networks in the numerical solution of linear integro-differential equations. *Applied Mathematics and Computation*, 188(1):427–432, 2007.

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

[78] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *Advances in neural information processing systems*, 27, 2014.

[79] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, 2020.

[80] J. Gopalakrishnan. Five lectures on DPG methods. *arXiv preprint arXiv:1306.0557*, 2013.

[81] D. Gottlieb and S. A. Orszag. Numerical analysis of Spectral Methods. *Philadelphia, Pennsylvania*, 19103, 1977.

[82] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, 2008.

[83] G. Gripenberg. Approximation by neural networks with a bounded number of nodes at each level. *Journal of approximation theory*, 122(2):260–266, 2003.

[84] P. Grohs, A. Jentzen, and D. Salimova. Deep neural network approximations for solutions of PDEs based on Monte Carlo algorithms. *Partial Differential Equations and Applications*, 3(4):Paper No. 45, 41, 2022.

[85] N. J. Guliyev and V. E. Ismailov. On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Networks*, 98:296–304, 2018.

[86] A. K. Gupta and S. Nadarajah. *Handbook of beta distribution and its applications.* CRC press, 2004.

[87] R. Haberman. *Applied Partial Differential Equations with Fourier series and Boundary Value Problems.* Pearson Higher Ed, 2012.

[88] D. W. Hahn and M. N. Ozisik. *Heat conduction.* John Wiley & Sons, 2012.

[89] B. Hanin. Universal function approximation by deep neural nets with bounded width and ReLU activations. *Mathematics*, 7(10):992, 2019.

[90] B. Hanin and M. Sellke. Approximating continuous functions by ReLU nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[91] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[92] D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[93] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[94] S. He, K. Reif, and R. Unbehauen. Multilayer neural networks for solving a class of partial differential equations. *Neural networks*, 13(3):385–396, 2000.

[95] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learningLecture 6eRMSprop: Divide the gradient by a running average of its recent magnitude, 2012. (Last accesed: August 2023).

[96] E. E. Holmes, M. A. Lewis, J. Banks, and R. Veit. Partial Differential Equations in Ecology: Spatial Interactions and Population Dynamics. *Ecology*, 75(1):17–29, 1994.

[97] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[98] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[99] B. Huang and J. Wang. Applications of physics-informed neural networks in power systems—A review. *IEEE Transactions on Power Systems*, 38(1):572–588, 2022.

[100] T. Hughes. The Finite Element Analysis, 1987.

[101] J. Hull. *Options, Futures, and Other Derivatives.* Pearson, 2015.

[102] F. P. Incropera, D. P. DeWitt, T. L. Bergman, A. S. Lavine, et al. *Fundamentals of heat and mass transfer*, volume 6. Wiley New York, 1996.

[103] V. Isakov. *Inverse problems for Partial Differential Equations*, volume 127. Springer, 2006.

[104] U. Iturrarán-Viveros and J. O. Parra. Artificial Neural Networks applied to estimate permeability, porosity and intrinsic attenuation using seismic attributes and well-log data. *Journal of Applied Geophysics*, 107:45–54, 2014.

[105] A. Ivakhnenko and V. Lapa. *Cybernetics and Forecasting Techniques.* Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

[106] J. D. Jackson. Classical electrodynamics, 1999.

[107] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[108] L. Jianyu, L. Siwei, Q. Yingjian, and H. Yaping. Numerical solution of differential equations by radial basis function neural networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 1, pages 773–777. IEEE, 2002.

[109] L. Jianyu, L. Siwei, Q. Yingjian, and H. Yaping. Numerical solution of elliptic partial differential equation using radial basis function neural networks. *Neural Networks*, 16(5-6):729–734, 2003.

[110] H. Jilani, A. Bahreininejad, and M. Ahmadi. Adaptive finite element mesh triangulation using self-organizing neural networks. *Advances in Engineering Software*, 40(11):1097–1103, 2009.

[111] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[112] R. Kachurovskii. Monotone operators and convex functionals. *Uspekhi Matematicheskikh Nauk*, 15(4):213–215, 1960.

[113] E. Kansa, H. Power, G. Fasshauer, and L. Ling. A volumetric integral radial basis function method for time-dependent partial differential equations. I. Formulation. *Engineering Analysis with Boundary Elements*, 28(10):1191–1206, 2004.

[114] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[115] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. VPINNs: Variational Physics-Informed Neural Networks for solving Partial Differential Equations. *arXiv preprint arXiv:1912.00873*, 2019.

[116] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[117] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[118] P. Kidger and T. Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

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

[120] S. Koroglu, P. Sergeant, and N. Umurkan. Comparison of analytical, finite element and neural network methods to study magnetic shielding. *Simulation Modelling Practice and Theory*, 18(2):206–216, 2010.

[121] M. Kumar and N. Yadav. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. *Computers & Mathematics with Applications*, 62(10):3796–3811, 2011.

[122] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider. A theoretical analysis of deep neural networks and parametric PDEs. *Constructive Approximation*, 55(1):73–125, 2022.

[123] D. La Torre, H. Kunze, F. Mendivil, M. Ruiz Galan, and R. Zaki. Inverse Problems: Theory and Application to Science and Engineering 2015. *Mathematical Problems in Engineering*, 2015:1–3, 2015.

[124] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[125] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[126] Z. K. Lawal, H. Yassin, D. T. C. Lai, and A. Che Idris. Physics-informed neural network (PINN) evolution and beyond: a systematic literature review and bibliometric analysis. *Big Data and Cognitive Computing*, 6(4):140, 2022.

[127] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

[128] G. Leobacher and F. Pillichshammer. *Introduction to quasi-Monte Carlo integration and applications*. Springer, 2014.

[129] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[130] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007.

[131] J. Li, J. Yue, W. Zhang, and W. Duan. The deep learning Galerkin method for the general Stokes equations. *Journal of Scientific Computing*, 93(1):5, 2022.

[132] X. Li and X. Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 4520–4524. IEEE, 2015.

[133] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, 2021.

[134] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[135] Y. Liao and P. Ming. Deep Nitsche Method: Deep Ritz Method with Essential Boundary Conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2021.

[136] H. Lin and S. Jegelka. ResNet with one-neuron hidden layers is a universal approximator. *Advances in neural information processing systems*, 31, 2018.

[137] M. Liu, Z. Cai, and J. Chen. Adaptive two-layer ReLU neural network: I. Best least-squares approximation. *Computers & Mathematics with Applications*, 113:34–44, 2022.

[138] D. L. Logan. *A First Course in the Finite Element Method.* Cengage Learning, 4th edition, 2010.

[139] Y. Lu, J. Lu, and M. Wang. A priori generalization analysis of the deep Ritz method for solving high dimensional elliptic partial differential equations. In *Conference on learning theory*, pages 3196–3241. PMLR, 2021.

[140] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

[141] M. Lysaker, A. Lundervold, and X.-C. Tai. Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time. *IEEE Transactions on Image Processing*, 12(12):1579–1590, 2003.

[142] N. Mai-Duy and T. Tran-Cong. Numerical solution of differential equations using multiquadric radial basis function networks. *Neural networks*, 14(2):185–199, 2001.

[143] N. Mai-Duy and T. Tran-Cong. Mesh-free radial basis function network methods with domain decomposition for approximation of functions and numerical solution of Poisson's equations. *Engineering Analysis with Boundary Elements*, 26(2):133–156, 2002.

[144] N. Mai-Duy and T. Tran-Cong. Approximation of function and its derivatives using radial basis function networks. *Applied Mathematical Modelling*, 27(3):197–220, 2003.

[145] N. Mai-Duy and T. Tran-Cong. Solving high-order partial differential equations with indirect radial basis function networks. *International Journal for Numerical Methods in Engineering*, 63(11):1636–1654, 2005.

[146] V. Maiorov and A. Pinkus. Lower bounds for approximation by MLP neural networks. *Neurocomputing*, 25(1-3):81–91, 1999.

[147] A. Malek and R. S. Beidokhti. Numerical solution for high order differential equations using a hybrid neural networkoptimization method. *Applied Mathematics and Computation*, 183(1):260–271, 2006.

[148] L. Manevitz, A. Bitar, and D. Givoli. Neural network time series forecasting of finite-element mesh adaptation. *Neurocomputing*, 63:447–463, 2005.

[149] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[150] C. C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 9(4):e1305, 2019.

[151] K. S. McFall and J. R. Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.

[152] D. L. McGee Jr. *Applications of neural networks to Partial Differential Equations.* PhD thesis, The University of Arizona, 1994.

[153] A. J. Meade Jr and A. A. Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9):19–44, 1994.

[154] A. J. Meade Jr and A. A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1–25, 1994.

[155] S. G. Mikhlin and L. Chambers. *Variational methods in mathematical physics*, volume 50. Pergamon Press Oxford, 1964.

[156] L. M. Milne-Thomson. *The calculus of finite differences.* American Mathematical Society, 2000.

[157] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[158] J. Moshfegh and M. N. Vouvakis. Direct solution of FEM models: Are sparse direct solvers the best strategy? In *2017 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, pages 1636–1638. IEEE, 2017.

[159] J. Muñoz-Matute, D. Pardo, and L. Demkowicz. Equivalence between the DPG method and the exponential integrators for linear parabolic problems. *Journal of Computational Physics*, 429:110016, 2021.

[160] J. D. Murray. *Mathematical Biology I.: An Introduction.* Springer, 2002.

[161] M. Nareklishvili, N. Polson, and V. Sokolov. Deep Partial Least Squares for Instrumental Variable Regression, 2023.

[162] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $\mathcal{O}(1/k^2)$. In *Dokl. Akad. Nauk. SSSR*, volume 269, page 543, 1983.

[163] M. E. Newman and G. T. Barkema. *Monte Carlo methods in statistical physics.* Clarendon Press, 1999.

[164] T. Nguyen-Thien and T. Tran-Cong. Approximation of functions and their derivatives: A neural network implementation with applications. *Applied Mathematical Modelling*, 23(9):687–704, 1999.

[165] J. T. Oden, A. Hawkins, and S. Prudhomme. General diffuse-interface theories and an approach to predictive tumor growth modeling. *Mathematical Models and Methods in Applied Sciences*, 20(03):477–517, 2010.

[166] Á. J. Omella, R. Celorrio, and D. Pardo. Sensitivity and uncertainty analysis by discontinuous Galerkin of lock-in thermography for crack characterization. *Computer Methods in Applied Mechanics and Engineering*, 373:113523, 2021.

[167] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett. Deep Learning Techniques for Inverse Problems in Imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020.

[168] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[169] D. Pardo. *Integration of hp-adaptivity with a two grid solver: applications to electromagnetics.* PhD thesis, University of Texas at Austin, 2004.

[170] J. Park and I. W. Sandberg. Approximation and radial-basis-function networks. *Neural computation*, 5(2):305–316, 1993.

[171] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[172] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[173] M. Penwarden, S. Zhe, A. Narayan, and R. M. Kirby. A metalearning approach for physics-informed neural networks (PINNs): Application to parameterized PDEs. *Journal of Computational Physics*, 477:111912, 2023.

[174] P. Petersen, M. Raslan, and F. Voigtlaender. Topological properties of the set of functions generated by neural networks of fixed size. *Foundations of computational mathematics*, 21:375–444, 2021.

[175] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8:143–195, 1999.

[176] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deep—but not shallow—networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[177] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[178] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[179] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[180] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, 2017.

[181] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations, 2017.

[182] M. Raissi, P. Perdikaris, and G. 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.

[183] P. Ramuhalli, L. Udpa, and S. S. Udpa. Finite-element neural networks for solving differential equations. *IEEE Transactions on Neural Networks*, 16(6):1381–1392, 2005.

[184] J. N. Reddy. *Introduction to the Finite Element Method.* McGraw-Hill Education, 2019.

[185] W. Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik. 1909.

[186] J. A. Rivera, J. M. Taylor, Á. J. Omella, and D. Pardo. On quadrature rules for solving Partial Differential Equations using Neural Networks. *Computer Methods in Applied Mechanics and Engineering*, 393:114710, 2022.

[187] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[188] S. Rojas, P. Maczuga, J. Muñoz-Matute, D. Pardo, and M. Paszynski. Robust Variational Physics-Informed Neural Networks. *arXiv preprint arXiv:2308.16910*, 2023.

[189] S. Rojas, I. Muga, and D. Pardo. A quadrature-free method for simulation and inversion of 1.5D direct current (DC) borehole measurements. *Computational Geosciences*, 20(6):1301–1318, 2016.

[190] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[191] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[192] S. Salsa. *Partial Differential Equations in Action: From Modelling to Theory*, volume 99. Springer, 2016.

[193] W. M. Saltzman. *Biomedical engineering: bridging medicine and technology.* Cambridge University Press, 2009.

[194] W. Samek, T. Wiegand, and K.-R. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[195] J. Schmidhuber. Annotated history of modern AI and Deep Learning. *arXiv preprint arXiv:2212.11279*, 2022.

[196] M. Shahriari, D. Pardo, A. Picon, A. Galdran, J. Del Ser, and C. Torres-Verdín. A deep learning approach to the inversion of borehole resistivity measurements. *Computational Geosciences*, 24(3):971–994, 2020.

[197] M. Shahriari, D. Pardo, J. A. Rivera, C. TorresVerdín, A. Picon, J. Del Ser, S. Ossandón, and V. M. Calo. Error control and loss functions for the deep learning inversion of borehole resistivity measurements. *International Journal for Numerical Methods in Engineering*, 122(6):1629–1657, 2021.

[198] Y. Shang, F. Wang, and J. Sun. Deep Petrov-Galerkin method for solving partial differential equations. *arXiv preprint arXiv:2201.12995*, 2022.

[199] Y. Shirvany, M. Hayati, and R. Moradian. Numerical solution of the nonlinear Schrodinger equation by feedforward neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 13(10):2132–2145, 2008.

[200] K. Shukla, P. C. Di Leoni, J. Blackshire, D. Sparkman, and G. E. Karniadakis. Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks. *Journal of Nondestructive Evaluation*, 39:1–20, 2020.

[201] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[202] N. Smaoui and S. Al-Enezi. Modelling the dynamics of nonlinear partial differential equations using neural networks. *Journal of Computational and Applied Mathematics*, 170(1):27–58, 2004.

[203] G. D. Smith. *Numerical solution of Partial Differential Equations: Finite Difference Methods.* Oxford University press, 1985.

[204] G. Smyrlis and V. Zisis. Local convergence of the steepest descent method in Hilbert spaces. *Journal of mathematical analysis and applications*, 300(2):436–453, 2004.

[205] I. N. Sneddon. *Elements of Partial Differential Equations.* Courier Corporation, 2006.

[206] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[207] W. A. Strauss. *Partial Differential Equations: An introduction.* John Wiley & Sons, 2007.

[208] T. Strouboulis, I. Babuška, and K. Copps. The design and analysis of the generalized finite element method. *Computer methods in applied mechanics and engineering*, 181(1-3):43–69, 2000.

[209] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning - Volume 28*, ICML'13, page III1139III1147. JMLR.org, 2013.

[210] A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, Jan. 2005.

[211] J. M. Taylor, M. Bastidas, D. Pardo, and I. Muga. Deep Fourier Residual method for solving time-harmonic Maxwell's equations, 2023.

[212] J. M. Taylor, D. Pardo, and I. Muga. A Deep Fourier Residual method for solving PDEs using Neural Networks. *Computer Methods in Applied Mechanics and Engineering*, 405:115850, 2023.

[213] A. N. Tikhonov and A. A. Samarskii. *Equations of Mathematical Physics.* Courier Corporation, 2013.

[214] I. Tsoulos, D. Gavrilis, and E. Glavas. Neural network construction and training using grammatical evolution. *Neurocomputing*, 72(1-3):269–277, 2008.

[215] I. G. Tsoulos, D. Gavrilis, and E. Glavas. Solving differential equations with constructed neural networks. *Neurocomputing*, 72(10-12):2385–2391, 2009.

[216] T. Uchiyama and N. Sonehara. Solving inverse problems in nonlinear PDEs by recurrent neural networks. In *IEEE International Conference on Neural Networks*, pages 99–102. IEEE, 1993.

[217] C. Uriarte, D. Pardo, I. Muga, and J. Muñoz-Matute. A Deep Double Ritz Method (D^{2}RM) for solving Partial Differential Equations using Neural Networks. *Computer Methods in Applied Mechanics and Engineering*, 405:115892, 2023.

[218] C. Uriarte, D. Pardo, and Á. J. Omella. A Finite Element based Deep Learning solver for parametric PDEs. *Computer Methods in Applied Mechanics and Engineering*, 391:114562, 2022.

[219] C. Uriarte, J. M. Taylor, D. Pardo, O. A. Rodríguez, and P. Vega. Memory-Based Monte Carlo Integration for Solving Partial Differential Equations Using Neural Networks. In *International Conference on Computational Science*, pages 509–516. Springer, 2023.

[220] M. V. Valueva, N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and computers in simulation*, 177:232–243, 2020.

[221] J. G. Van Bladel. *Electromagnetic fields*, volume 19. John Wiley & Sons, 2007.

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

[223] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education, 2007.

[224] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, 17(3):261–272, 2020.

[225] S. Weinzierl. Introduction to Monte Carlo methods. *arXiv preprint hep-ph/0006269*, 2000.

[226] S. Wojtowytsch and E. Weinan. Can shallow neural networks beat the curse of dimensionality? A mean field training perspective. *IEEE Transactions on Artificial Intelligence*, 1(2):121–129, 2020.

[227] N. Yadav, A. Yadav, M. Kumar, et al. *An introduction to neural network methods for differential equations*, volume 1. Springer, 2015.

[228] F. Yaman, V. G. Yakhno, and R. Potthast. Recent Theory and Applications on Inverse Problems. *Mathematical Problems in Engineering*, 2013:303154, 2013.

[229] L. Yang, X. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[230] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017.

[231] J. C. Ye, Y. Han, and E. Cha. Deep Convolutional Framelets: A General Deep Learning Framework for Inverse Problems. *SIAM Journal on Imaging Sciences*, 11(2):991–1048, 2018.

[232] Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

[233] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[234] L. Ziemiański. Hybrid neural network/finite element modelling of wave propagation in infinite domains. *Computers & structures*, 81(8-11):1099–1109, 2003.