Deep Learning for PDEs

Report of the joint APSC-NAPDE courses project

Paolo Joseph Baioni*

March 24, 2020

*paolojoseph.baioni@mail.polimi.it

Acknowledgements

Contents

Introduction **3**

1 Neural Networks and Deep Learning **5**
1.1 Architecture and operation of a Deep Neural Network 5
1.2 Further insights 5

2 Application to PDEs **6**
2.1 The Poisson-Dirichlet problem 6
2.2 Some more in-depth theoretical results 6

3 Implementation of neural-net **7**
3.1 Directory tree 7
3.2 The Neural Network class 8
3.3 The Optimizers class templates 8
3.4 Write_set and data 8
3.5 The main 8

4 Examples **9**
4.1 Building instructions 9
4.2 Reconstruction of a wave packet 9

Conclusions and further developments **10**

Appendix **11**

Bibliography **12**

Introduction

The numerical solution of partial derivative equations (PDEs) plays a fundamental role in applied mathematics, science and engineering.
The recent advances in machine learning (ML) and the successes obtained by the application of these techniques in various areas suggest the possibility of using ML in solving PDEs. This approach has considerable potential compared to other numerical methods: from the possibility of writing mesh-free algorithms to that of solving data-learned equations, whose explicit functional form is unknown; however the theory is not yet developed and consequently important results of convergence and stability are missing, as well as general rules that would allow to identify the optimal parameters for the design of numerical codes. Finally, some intrinsic characteristics of the method make it considerable as a tool which is complementary to traditional numerical methods, finding application in the field of real-time control.
The aim of this project is to get into deep learning techniques and study their possible applications to PDEs, also reporting some useful theoretical results, as a complement to the methods illustrated during the NAPDE course, and to implement from scratch a Numerical Analysis relevant Neural Networks based C++ program, so to both gain and verify a low-level, detailed and complete understanding of the method and to experiment on some of the programming techniques that have been deepened during the APSC and *"Strumenti di sviluppo e distribuzione di software per la ricerca scientifica"* courses held at PoliMi.
The structure of the report is as follows.
In chapter 1 we present the general architecture of a Deep Neural Network (DNN) and the main idea of functioning of the algorithm that allows it to learn from the data (Deep Learning), consisting of two phases, called *forward propagation* and *backward propagation*. Therefore, some sector-specific issues are considered in detail, such as: distinction between train, development and test sets, problems related to overfitting, possible regularization techniques aimed at reducing it, different optimization algorithms and an overview of the main parameters that must be tuned adequately to get good results.

In chapter 2 two possible approaches to solving PDEs via DNNs are exposed, also highlighting some choices that can be made in the formulation of the problem and in treating the boundary conditions. We then deepen the comparison between DNNs and the piecewise continuous linear function which are used as bases of finite element spaces of order one, in order to provide a greater intuition of the reasons why the DNN-based method works and to identify, albeit in this particular case, some general indications on the ideal number of nodes and layers of the DNN.
In chapter 3 we explain the programming architectural choices and the structure of the developed code, freely available on GitHub[1], as well as possible extensions.
In chapter 4, after providing instructions on how to compile, link and run the program, we show some examples by means of benchmark cases.
In appendix we report the [FreeFem++] code written for the computations performed in section 2.2.

[1] *https://github.com/pjbaioni/neural-net*

Chapter 1

Neural Networks and Deep Learning

1.1 Architecture and operation of a Deep Neural Network

1.2 Further insights

Chapter 2

Application to PDEs

2.1 The Poisson-Dirichlet problem

2.2 Some more in-depth theoretical results

Chapter 3

Implementation of neural-net

In this chapter an in-depth view of a Deep Learning C++ program, called *neural-net*, is given.
The program has been built using the [Git] distributed version control system, with GitHub as hosting service, and can be freely cloned or downloaded from the author's page: *https://github.com/pjbaioni/neural-net*.
In order to build and run the program, some not included dependencies are needed, in particular a C++ compiler, [Make], [gnuplot], [Eigen] and [Boost] libraries, while to build the documentation a TeX compiler has to be used; more details are given in the README.md file and in chapter 4, where it is shown how to build the program and run an example.

3.1 Directory tree

Downloading the repository "neural-net" one's find:

- the *data* folder, which contains the input and output data in .dat and .pot (for [GetPot]) format, [gnuplot] scripts in .gnu format, and their graphical output in .png format;
- the *doc* folder, containing this documentation in .tex and .pdf format, plus the *img* sub-folder where there are the images included by the TeX file;
- the *include* folder, containing the headers *GetPot.hpp*, an utility used for parsing parameters file and command line option, *gnuplot-iostream.hpp*, an utility used to output a graphical representation of the results in an interactive way, *NeuralNetwork.hpp*, which holds the NeuralNetwork **class**, and *Optimizers.hpp*, where all the optimizers employable from NeuralNetwork are defined as **class templates**;

- the *src* folder, which contain the source files main.cpp, NeuralNetwork.cpp, where the NeuralNetwork class member functions are defined, the Makefile which can be used to automatic build the main program, and the *write_set* sub-folder, where the *write_set.cpp* file used to generate datasets is placed;
- the COPYING file, a plain text file containing copying informations and licenses;
- the README.md file, a markdown file containing basic informations;
- the (hidden) *.gitignore* file, that trace the file extensions which are not being pushed to the remote, mainly compilation files, executables and comments file.

Despite the different extensions, every non-png nor non-pdf file can be opened by any text editor.

3.2 The Neural Network class

3.3 The Optimizers class templates

3.4 Write_set and data

3.5 The main

Chapter 4

Examples

4.1 Building instructions

4.2 Reconstruction of a wave packet

Conclusions and further developments

Appendix

Bibliography

[APSC] Advanced Programming for Scientific Computing course lectures, Politecnico di Milano, 2019.

[Andrew NG] Professor Andrew NG's Deep Learning MOOC at Coursera, *https://www.coursera.org/specializations/deep-learning*, consulted in 2019.

[Boost] Boost C++ libraries, *https://www.boost.org/*

[cppreference.com] An online reference of the C++ language, *https://en.cppreference.com/w/cpp*

[Eigen] Eigen C++ template library for linear algebra, *https://eigen.tuxfamily.org/*

[FreeFem++] FreeFem++ PDE solver, *https://freefem.org/*

[GetPot] GetPot command line parser, *http://getpot.sourceforge.net/*

[Git] Git distributed version control system, *https://git-scm.com/*

[gnuplot] A GNU graphing utility, *http://www.gnuplot.info/*

[gnuplot-iostream] A C++ interface to gnuplot, *https://github.com/dstahlke/gnuplot-iostream*

[Jinchao Xu et al.] Juncai He, Lin li, Jinchao Xu, Chunyue Zheng, *ReLU Deep Neural Networks and Linear Finite Elements*, 2018, found at *https://arxiv.org/abs/1807.03973*

[Kailai et al.] Kailai Xu, Bella Shi, Shuyi Yin, code and technical report of the project for the course CS230 *Deep Learning, Winter 2018, Stanford University*, found at *https://github.com/kailaix/nnpde*

[Kingma-LeiBa] Diederik P. Kingma, Jimmy Lei Ba,*Adam: a method for stochastic optimization*, 2015, found at *https://arxiv.org/abs/1412.6980*

[Make] GNU Make doc at *https://www.gnu.org/software/make/manual*

[Weinan-Bing] Weinan E, Bing Yu *The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, 2017, found at *https://arxiv.org/abs/1710.00211*