Mathematical Inquiry I: Module 2

Lecture notes

MATH 2030

Fall 2024

Dr. Alex Bihlo

Department of Mathematics and Statistics

Memorial University of Newfoundland

A1C 5N3 St. John's (NL), Canada

abihlo@mun.ca

Version: October 1, 2024

Preface

The goal of this course is for you to learn the essential elements of writing technical text, using the LaTeX typesetting language. This course has been designed to be completed asynchronously, and these lecture notes should provide some brief background to the problems that will be considered in this course. For many students the information provided in these notes as well as some guided self-study will be enough to complete the projects outlined here. For others, some more information may be required, and for those I suggest to consult myself as well as the references provided in this text.

St. John's, September 2024

Contents

1 Introduction **iii**
1.1 Motivation . . . iii
1.2 Resources . . . iv
1.2.1 LaTeX . . . iv
1.2.2 `Python` . . . iv
1.3 Style guidelines for your reports . . . v
1.4 References . . . vii
1.5 Use of generative AI . . . viii
1.6 What to include in your report . . . ix
1.7 Submitting your report . . . x
1.8 Marking . . . xi

2 Project 2: A study on neural network optimizers **1**
2.1 Introduction . . . 1
2.2 Project . . . 8
2.2.1 A study on neural network optimizers in 1D. . . . 8
2.2.2 A study on neural network optimizers in 2D. . . . 9
2.2.3 Training a neural network . . . 10
2.3 Why this project is relevant . . . 11
2.4 Further reading . . . 11

References **12**

Chapter 1

Introduction

1.1 Motivation

The aim of this course is to teach you technical writing in the mathematical sciences. Knowing how to write high-quality technical reports is of practical relevance whether you want to become a researcher or work in industry.

There are two parts to writing a technical report, doing research work worthy of being reported, and then preparing the actual report itself. In this course you will do both. You will be working on three independent projects and will then write technical reports on the findings you obtained.

The three projects are practical and contain aspects you most likely will encounter in practice in your career after graduating. These projects are designed to challenge you, in that they introduce problems that you may have never seen before. They will introduce mathematical concepts you most likely will have never seen before. While this may seem intimidating, solving new problems and being able to figure out how to use mathematical and computational tools you have never used before is probably your most valuable skill as a mathematician. Employers will hire you because you possess this skill.

Working on problems that go beyond your current mathematical understanding is an important exercise you should aim to practice continuously during your undergraduate program. This will allow you to gain experience that is immensely valuable both for academia (should you plan to go to graduate school) and private-sector industry.

To guide your exploration on the projects to be tackled in this course, for most projects a concise introduction to these problems will be provided. This introduction should be enough to get your started but may not go deep enough for your personal taste. Therefore, some key references will be provided where further information could be found. Those references should mostly be regarded as a starting point for your own literature review, they may not be the references that will be most helpful for everyone. Some students may prefer classical textbooks, other may prefer blog posts on the internet explaining the topics covered in these projects. Learning to do a proper literature search is an important part of the scientific writing experience.

Regardless of the problem you are working on, changes are high that someone else has worked on the exact same problem before your. Chances are even higher that someone else has worked a similar problem and used methods that will be helpful for your specific problem. As such it is best practice in scientific research to begin with a literature review.

There are many ways how a scientific literature research can be conducted but one standard

tool that is frequently used is *Google Scholar*,

https://scholar.google.com/

where you can search for scientific papers and books on every subject imaginable.

1.2 Resources

For this course, no previous knowledge of LaTeX or `Python` is assumed. Below I list some of the resources that should help you getting started with both.

1.2.1 LaTeX

A short introduction to the fundamentals of LaTeX is provided as slides and lecture recording. In addition to these resources, there are many great tutorials on LaTeX available online. A simple tutorial covering similar material as the slides provided would be *Learn LaTeX in 30 minutes*:

https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes,

which is available on the website of the online LaTeX editor Overleaf. A more detailed introduction is Chapter 3 of the MATH 2130 manual available on our Department of Mathematics and Statistics website:

https://www.math.mun.ca/~m2130/Manual/ch3_typesetting.pdf.

To use LaTeX you will first have to either install it locally on your computer, with versions available for Linux, Windows and MacOS, or use an online service like Overleaf, which is free for single user. For installation guides see:

https://www.latex-project.org/get/.

While installing LaTeX is generally straightforward, should you run into any issues related to your specific hardware that cannot be resolved easily, I would suggest using Overleaf instead. Note that if you install LaTeX on your own computer you may also have to install a suitable LaTeX editor.

1.2.2 `Python`

`Python` is a high-level, interpretative programming language that has emerged as one of the standard tools in scientific computing, data analysis and machine learning. While for the purpose of this course we will use `Python` (version 3) mainly for simple short programs and visualization purposes, I strongly suggest that you make every effort to pick up as much of this language as you can. Most of the jobs available for applied mathematician in industry require solid experience in `Python` and the more often you use it for the various courses in your undergraduate program, the more experience you will gain using it.

A short (interactive) introduction to `Python` is provided for this course, both as a *jupyter notebook* and as a video recording. In addition to this introduction by me, as with LaTeX there

are a multitude of tutorials and online resources (including free online courses, for example on YouTube) available that will allow you to familiarize yourself with the basic functioning of `Python`. One short resource to start with would be *A Byte of Python*:

https://python.swaroopch.com/.

There are a great many packages available which considerably extend the base functionality of `Python`. In the applied mathematics, scientific computing and data science setting, the most important packages to get familiar with are *Numpy*, *Scipy*, *Pandas*, *Matplotlib*, *Plotly* and *Scikit-Learn*. In this course we will use only some elementary aspects of these packages, so it will be enough if you familiarize yourself with the main purpose of these extensions rather than trying to understand them exhaustively (which would be an overwhelming undertaking).

`Python` can be installed on Linux, Windows and MacOS. As with LaTeX there are also online `Python` interpreter available. One particularly simple and convenient way to use `Python` is via *jupyter notebooks*:

https://jupyter.org/.

`Jupyter notebooks` allow mixing text with code and thus allow for a neat and interactive way to run and present `Python` code and its associated results. I strongly advise to use `jupyter notebooks` for this course as a means to provide the code for the projects to accompany your written reports.

Personally I use `Google Colab` (which is free):

https://colab.research.google.com/,

for most of my `Python` programming, which combines the convenience of `jupyter notebooks` with the ability to interact with local files saved on Google Drive, exporting notebooks to `Github`, etc. It also comes with essentially all `Python` extensions pre-installed that are needed to do most projects in applied mathematics, scientific computing and data science.

Should you run into any issues installing `Python` or any of the associated packages on your local computer (which is fairly straightforward on Linux and MacOS, but slightly less straightforward on Windows), or if you encounter severe performance issues (maybe due to having an outdated computer), then I strongly advise to use an online `Python` interpreter such as `Google Colab` instead.

1.3 Style guidelines for your reports

A template for a report (along with the LaTeX source code) can be found on the course website on Brightspace. It follows the typical style of essentially all papers in the mathematical sciences:

1. *Title*: Choose a meaningful (but typically short) title that gives the reader a reason to look at your work.
2. *Authors:* The list of authors who contributed to the paper in an essential way, along with their work addresses and contact information (for you this is just your own name, since all projects are to be completed by yourself).

3. *Abstract*: This is a very short summary of the main findings of your work. The abstract would normally be the first part read by a prospective reader so be sure to make it advertising and easily readable for the reader to be keen on reading your paper in its entirety.
4. *Introduction*: The introduction would usually contain a broader, mostly non-technical discussion of the problem you are considering in your paper. This is typically where you would include the literature review for your work.
5. *Methods*: Here you would begin to delve into the technical details of your work, e.g. explaining the mathematical or computational methods used.
6. *Results*: Applying the methods described in the previous section, this would be the place to include your results, including tables, figures, etc.
7. *Conclusion*: Here you would summarize the results of your paper and could provide some outlook of what could be considered as next steps in the wider context of the work in which your paper is situated.
8. *Acknowledgements*: Here you would thank people that have helped you in some way with your work (e.g. any peer-reviewers, colleagues, etc.), and acknowledge funding from sponsors.
9. *Appendices*: This is optional, but could include the source codes used in your work, longer proofs to theorems, etc.

The above is a general formula, and not all papers have to follow the exact same formula. Shorter papers may combine the *Methods* and *Results* section, and this may also be appropriate for some of your own reports for this course. Many papers within the area of pure math also do not separate the *Methods* and the *Results* sections, since the main aim of such papers is to prove one or more statements so there is no natural separation between the methods used and the results obtained.

Besides the above general structure for a mathematical report, the following is a short collection of best practices that you should keep in mind when writing your reports:

1. Technical reports are typically written in neutral, concise language; exaggerations should be avoided.
2. Technical reports are more often written in Third Person ("We show that") rather than in First Person ("I show that"), even if there is only a single author.
3. Despite the language in technical reports being neutral, is has to follow all the regular rules of the English language.
4. Your task is to back up everything you are reporting. Sweeping statements without proofs have to be avoided.
5. If you have a conjecture about a statement that you cannot back up, it is acceptable to include this as a conjecture, and explain your reasoning behind it, and what would be required to exhaustively prove this statement.
6. Including plots and tables can make a report more easily digestible, providing summarizing information in an accessible way. For this to work, plots and tables should be as self-contained as possible: Axes have to be labelled; multiple curves should have a legend; if plots are in color, think if they would be still understandable if printed in black and white; if numbers are reported then explain the units being used.

7. If you use information taken from books or scientific articles it is imperative to cite them.

As with any kind of writing, also scientific writing requires a lot of practice. Do not get discouraged if you find it difficult or overwhelming to getting started! For more suggestions, consult the MATH 2130 course manual or [5].

1.4 References

A hallmark of a technical report is that is includes references to other technical documents, such as papers, technical reports or books. As you research a scientific problem, you would start with an extensive literature review. There is not much worse for a scientist than to write a paper on a topic only to discover later that the exact same problem has already been treated elsewhere. To avoid such an unfortunate situation, knowing your subject area is key.

While for the present course some background information on the problems to be considered is already given in these lecture notes, a good report will expand on the problems you will be working on. Doing further literature research, reading papers and appropriate sections of books will strengthen the report you will be writing.

It is crucial to stress the importance of proper citations of sources used. If you find a useful statement for your report in a paper, then you have to cite that paper, and make it part of the bibliography of your report. LaTeX provides a suitable reference management system in the form of `BibTeX`, which allows you to easily add references into a database which can then be used within LaTeX.

There are many different citation styles, but the most common in the mathematical sciences is to either cite a paper by its reference number of the bibliography (as is done in these lecture notes) or in the form of Authors/Year. To cite the book by S.G. Krantz you thus could either use the numerical form, that is [5], or the Author/Year form, that is (Krantz, 2017). The style of your bibliography can be set globally in your LaTeX document. The ordering of the bibliography can be customized as well, with alphabetic ordering being the most common (another option would be to order references according to their occurrence in your report, meaning the first paper you cite would be [1] in your bibliography, the second paper [2], etc.).

Independent of the citation style you are using, references are important. It is not a weakness to cite many papers in a report, as long as the references you cite are relevant to your work. In fact, unless a statement is within the domain of common knowledge (such as the Pythagorean Theorem, the rules of calculus, etc.), it has to be cited.

The source from which you cite is also important. Books and peer-reviewed scientific papers are more credible sources than a blog post off the internet, which usually has not been peer-reviewed and which may be taken down at any point. It is of course absolutely allowed to use information from the internet and cite it accordingly (just indicated what date you accessed that information), but actual research papers largely avoid material from the internet, simply because most of the scientifically useful material can be found in more credible sources. Please consider this for your own report and avoid using pages off the internet as your sole source of information.

As an example, when you are working on Project 2, and you want to describe gradient descent in some more detail, do not cite the Wikipedia article on gradient descent. There are several original papers describing popular gradient descent based optimizers, such as *Adam*, that would be a great source [4]. If that information is too hard to digest for you, try finding an introductory book on machine learning which shortly describes the most important optimization algorithms

and provides some useful background information for you to include in your report. According to Google Scholar, the paper by Kingma and Ba on the Adam optimizer has been cited 191,320 (!) times as of late September 2024, so there is an abundance of information on this (and similar) method(s) out there that will be accessible to every skill level.

Using information without properly citing the relevant sources (independent of whether these sources are books, articles or the internet) is plagiarism, which is a serious academic offence, both in science and for this course. Avoid plagiarism at any cost!

Any task completed that involves plagiarized code/text will receive zero points on that task.

1.5 Use of generative AI

Recent years have seen the proliferation for numerous generative AI tools, such as *image generators*, including Dall-E, Midjourney, and Stable Diffusion, and *large-language models*, including Claude, GPT, PaLM, LLaMa, and their chat-bot interfaces including Bing chat, chatGPT, and Bard. These tools have also found their way into academia, with many major publishing houses now mandating disclosure of the use of generative AI in research publications.

Generative AI particularly impacts this course, as it allows the generation of images and large bodies of text from simple textual prompts. While the legality (and ethics) around these generative AI tools still largely remains in question, with regulatory frameworks currently being developed, it is hard to imagine that the importance of these tools will diminish within the next few years. As such, this course takes a pragmatic standpoint towards the use of generative AI:

You are allowed to use generative AI provided you declare that you used it.

That is, uses of generative AI in writing your project report have to be disclosed by including a *Declaration on the use of generative AI in the writing process* section in your report. In this declaration you have to clearly indicate which tools were used for which parts of your report, and for what reasons. For example, if you used chatGPT to help you write an introduction to your report, then your declaration on the use of generative AI should include a statement such as "I have used chatGPT for providing a draft for the introduction of this report, which I have then reviewed and edited.". This statement can be either in the beginning (after the abstract) or in the end (before the references) of your report.

Please pay attention that it is never advisable to simply take the output of large-language models at face value, as they typically require extensive fact checking. This is particularly true in any technical area, as the limited amount of training data in these areas often leads large-language models to hallucinate, i.e. to simply invent incorrect (but often convincingly sounding) text.

Note that you retain full responsibility for the use of any text (or images) created from generative AI, and as such will receive point deductions for incorrect statements in your reports, even if those mistakes were made by generative AI. If you use the outputs of generative AI for more than two reports, or if you do not disclose their use, this will be treated as an academic offence comparable to plagiarism. Undisclosed use of generative AI will thus automatically incur zero points on all tasks for which these tools have been used.

1.6 What to include in your report

In Section 1.3 I have provided a short possible skeleton for your report. What precisely to include in each section is largely up to you. Here are some general thoughts you might want to consider:

1. *Introduction:* A good report should be self-contained, i.e. it should be readable by a colleague who has a solid mathematical education but who would not necessarily be an expert in the specifics of the report. For Project 2 you could provide an overview of numerical optimization in general, why neural networks are important, etc. Your goal for the introduction would be to convince the reader that what you are presenting in your report is an important problem that needs to be studied further.
2. *Methods:* Here you would explain why you chose the methods you were using. You would explain the model you were considering in such a manner that it would be readable to someone who has never seen that model before. While for the purpose of this course the models and methods are mostly given (e.g. implementing gradient descent for Project 2), you would still provide arguments as to why this is the right choice. You could also discuss competing or more general approaches (as applicable), highlight potential limitations, explain why you believe your approach was appropriate, or how these limitations would restrict the generality of what you were doing.
3. *Results:* Here you present a selection of your graphs, tables, etc. An interpretation to all these results has to be given. That is, it is not enough to just include the plots you were asked to produce without any further explanation. Give a critical analysis, explain what these plots are showing, how they are solving the problems you set out to solve, where they fall short, etc.
4. *Conclusion:* Give a concise summary of what you have accomplished in your research and why it is relevant. Honestly assess the strengths and weaknesses, e.g. where further research would be needed to corroborate your hypotheses. Shortly describe what you think would be appropriate to do next within the wider area of that project. Would more complicated models have to be considered? Would you need more data to make more justified statements about that research problem?
5. *Appendix:* Here you could provide (parts of) the `Python` codes you have written. If your code is very long then it would not be appropriate to include all of it as this will make your report appear rather messy. In particular, the various `import` statements, variable initializations, etc. can usually be omitted; rather, you could select and discuss some of the key routines you have written, in particular if they clarify other parts of your report. In other words, if you describe your methods in the *Methods* section, you could reference your computational routines in the *Appendix* for clarification purposes.

There is no general rule as to how to structure a research paper, and how much weight to assign to each section. Different colleagues will have different opinions, and ideally you will find your own style that will work for you. Aim for a paper that is interesting to read, factually correct, and that does a proper job in convincing the reader that what you have done is of scientific value and should be considered further.

The goal of research is to produce papers that will be read and cited by colleagues. Writing a paper that nobody wants to cite is frustrating for the authors. While most research is highly

specialized and will not gather as many citations as the aforementioned paper by Kingma and Ba, it is still the case that the presentation of your research results is a main contributor of how your paper will be received by the scientific community. You may have proven an important statement or obtained an important result but if you present the proof or that result in an incomprehensible or sloppy way, riddled with typos and grammatical mistakes, then chances are high that your paper will not be successful.

Please remember this for your reports as well, *the presentation of your results is just as important as the results themselves.*

Remark 1. The length of a report is **not** an indication for the quality of a report. A concise, well-thought out 2 page report will be better than an unstructured and unorganized 10 page report. As everybody has a different style, I will not provide guidelines on the lengths of your reports!

Remark 2. In computational mathematics it fortunately becomes more and more customary to provide the source codes for the research you have done. This was unfortunately not the case in the past, which made it hard for reviewers to assess the correctness of the results reported in a scientific paper. The source code can be provided in various ways, e.g. making it publicly available in repositories on online code hosting services such as `Github`, on the website of the journal, or in the appendix of your paper itself. Here we will follow best practices so I ask you to submit your codes along with your reports. Short codes can be provided in the appendix of your paper, longer codes could be uploaded as `.py` (`Python`) or `.ipynb` (`Jupyter notebook`) files to the respective project Dropbox.

1.7 Submitting your report

When you try to publish a scientific paper that you have written in a scientific journal, it will have to undergo *peer-review*. Here the editor of the journal you are submitting your work to will select a few experts in the field of the article and will ask them to carefully read your paper and provide reports on it. Based on these reports, your paper will either be *accepted*, has to *undergo a revision*, or will be *rejected*.

We will use the peer-reviewed method for assessing your reports as well. Once you are happy with your report (or, at the latest, at the deadline for each project) you will send it to me as the 'editor' (in practice you just upload it to the assignment Dropbox on Brightspace). I will then send your report to two of your colleagues and ask them to write a short critique on your report. This critique should honestly (but politely!) assess the strengths and weaknesses of your report. Note that peer-reviews for journals are anonymous, so you will not know who will be reviewing your work, and the reviewers should not include their names in their reports.

Learning to write a report as a reviewer for a scientific document is an important skill as well. In practice you come across a variety of reviewers and not all of them are friendly and polite, and unfortunately not all of their reports are really useful for you as author at all. Here we aim to learn best practices of being a supportive reviewer, with the goal of improving your peers' reports. There will be no *rejection* option, but your goal as reviewer will be to find as many weaknesses as possible in the report you are reviewing, along with concrete suggestions for improvements.

As a reviewer, you can also go through the list of best practices provided in Section 1.3. Have these best practices been followed? If not, then you could provide some helpful suggestions on

how the report could be improved. Are the results faulty? Are the arguments hand-wavy? Are the conclusions justified? Is the presentation of results understandable? Is enough background information provided?

Once your review report is done, you will send it back to me (again, there will be a Dropbox where you can upload the report), and I will then forward this report to the author of the paper you reviewed. The task of you as the author is then to incorporate the feedback you have received. You would correct any mistakes found by the reviewers, or, if you do not agree with a reviewer on some of his/her remarks, you would provide an argumentation as to why you did not incorporate these remarks.

The correction process thus consists of two steps: You correct your paper according to the suggestions of the reviewers, and you collect all of your corrections in a response document (usually entitled *Response to the reviewers*). To give an example, say your reviewer remarks that you forgot to label some axes in your plots, you would then (i) make new plots with the proper axes labels for your paper itself, and (ii) write in your response document that you have included these new plots. Practically, this could be done by copying the respective remark from the reviewer's report in your response document and providing your response thereafter, e.g.:

Remark by reviewer: "I should also like to note that in Figure 2 the x-axis has not been labelled."

Response: "We thank the reviewer for catching this issue. We have added a proper label to this Figure. The x-axis now correctly identifies this variable as *time*."

If there are multiple issues being raised by the reviewer it is not necessary to thank them for each and every single point. Still, try to maintain an overall grateful tone in your response document, even if you disagree with what the reviewer has been proposing. Staying polite despite having an unfriendly reviewer is a skill that unfortunately has to be honed in science (as well as in industry).

In practice, the editor would then forward your response document along with the corrected version of your paper to the same reviewers again, who then will make a final decision (or require some more modifications). While it is generally the case that there will be only one revision, some reviewers may require multiple back-and-forth until they will come to a final decision on whether your paper can be accepted or has to be rejected.

Here we will not do multiple rounds of revision. You will incorporate the feedback of your reviewers within **one week** and send the final version of your paper to me (in practice you will re-upload it to the Dropbox, along with your response document), and I will not send it back to the reviewers. This once-revised version of the paper is what I will then be grading.

As with writing a technical paper itself, also writing reviews and understanding the intricacies of the peer-review process is a skill that takes some time and practice to acquire; upon completing this course, you will have a better understanding of writing reports, reviewing and revising them.

1.8 Marking

The total marks for project 2 are as follows:

1. Project 2: 33% (25% for your own report + 8% for your peer-review reports)

I will grade your report only *after* it has been peer-reviewed by your fellow colleagues, and corrected by yourself. That is, you may have made a mistake in one of your sections and your reviewer catches this mistake. You then prepare a revision for your report upon which your work

is improved, and I will only grade the improved version (along with the response document), not the original version.

Note though that this is **not** a loophole for gaining extra time for submitting your project report. You cannot submit an incomplete project report for peer-review and then complete that project report in the course of the peer-review corrections. The purpose of peer-review is to assess a scientific document which you regard as publishable. You would not send a scientific paper to a peer-reviewed journal and have a section missing, in the hopes that the reviewers will tell you what to write for that section; that is not the task of the reviewers, and such a paper would be rejected by the editor before sending it out to peer-review! Therefore, you have to submit a complete project report at the initial submission deadline, and the reviewers' job is to assess the strengths and weaknesses of that report. They may require some extra work (e.g. saying that an extra figure or some more detailed description would be helpful), but this would be to improve your paper further, rather than because your paper was incomplete.

To reward the effort you put into peer-reviewing your colleagues' report, part of your grade will go towards the quality of your review report. Note that the amount of marks you obtain for peer-reviewing will be independent of the quality of the project report you are reviewing. If you review a report that is already excellent, then you can write a justified report as to why this report is excellent, i.e. why it should be published "as is" and does not have to be revised. If the report you review has weaknesses then you write a justified report suggesting how to improve those weaknesses, i.e. why it should be revised and in what way.

In either case, the marks you obtain for your reviewing work will depend on the quality of your review report alone, not on the quality of the report you are reviewing. Expect to review on two reports for project 2.

Chapter 2

Project 2: A study on neural network optimizers

2.1 Introduction

Deep learning is a sub-field of *machine learning*, which itself is a sub-field of *artificial intelligence*. While this first project aims to give you a glimpse into the field of deep learning, for the sake of context it makes sense to first position deep learning in the wider context of machine learning and artificial intelligence. So let us begin with a few definitions.

Definition 1 (Artificial intelligence)**.** Artificial intelligence is the effort to automate intellectual tasks normally performed by humans.

This is a rather wide definition, and encompasses examples such as chess programs, expert systems, self-driving cars, image and speech recognition, and autonomous robots.

Definition 2 (Machine learning)**.** The capability of a computer (or, more generally, of an AI system) to acquire their own knowledge by extracting patterns from raw data.

This definition is already narrower than the previous definition on artificial intelligence. It excludes, for example, classical chess programs, which rely on hard-coded rules, rather than extracting knowledge of the game from the game itself (although this can be done using, for example, *reinforcement learning*, which we will visit in the last part of this course).

Problems that are solved with machine learning from the previous list still include self-driving cars, image and speech recognition, and autonomous robots.

Before we provide an introduction to deep learning, it makes sense to highlight what distinguishes machine learning from other tasks in (computer) science. Fig. 2.1 highlights the main difference between a classical approach to science and the machine learning approach to science.

(a) Classical approach to (computer) science. (b) Machine learning approach.

Figure 2.1: Machine learning vs. the classical scientific method.

In classical science we usually begin with some observations (data) of some processes that we would like to understand, and devise some rules (a theory) that we believe this data is

following. Then we test our theory by comparing the answers our theory gives to the data we have collected. In machine learning we reverse this direction, by providing the data along with some characteristics of that data, and aim to learn the rules that this data is obeying.

Having provided a high-level overview of what artificial intelligence and machine learning are, we now have to address the question of how machine learning actually works in practice. Fundamentally, all machine learning systems (including deep learning based ones) learn by finding useful representations of the input data and a transformation to the output data.

As a simple example, let us consider the task of classifying data points into two classes, a so-called *binary classification problem*. Concretely, consider the data given in Figure 2.2.

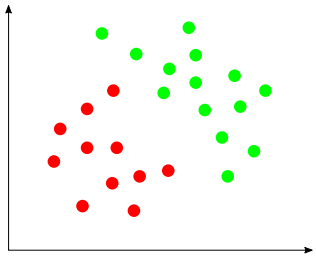

Figure 2.2: A dataset for binary classification.

What would be a simple rule to classify whether data falls into the red or the green class? One straightforward idea would be to separate the data as is by a straight line, and then determine that every data point below (above) the straight line is of class red (green), see Fig. 2.3.

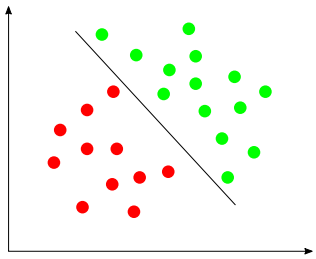

Figure 2.3: A machine learning model for this dataset.

Knowing this line is particularly useful as suppose we were to add another example (i.e. another point) to our dataset, then we would like to know whether this example will likely correspond to the red class or to the green class. The decision would then be based on whether it falls above or below the straight line we found.

What we have briefly described in the paragraph above is the main paradigm of machine learning: We train a machine learning model based on some given *training data*, which in our case was the collection of two-dimensional data points with an associated label to it (either red or green). Once we have trained a model, the hope is that the model would perform equally well on *unseen* data; that is, if we would draw other examples, constituting the *testing data*, from the same underlying data distribution that generated our training data, then our model should *generalize* well to this new data, meaning it should yield correct predictions of the labels (red or green) of the new data as well.

In summary, the data shown in Fig. 2.2 constitutes our training data, the line found in Fig. 2.3 is the machine learning model, and any data not included in the training data that we would present our model will be classified based on this model.

The question remains whether the model we have found in Fig. 2.3 is the best possible model. What characterizes this model is that we need to check both the x- and the y-coordinate of any new data point that we would add and compare it against the associated x- and y-component of the separating line.

Another solution would be to first *rotate* the data, and then classify it, giving the result shown in Fig. 2.4. This model is simpler as we now just have to check the x-component of each data sample to determine whether or not it will be a red or a green sample.

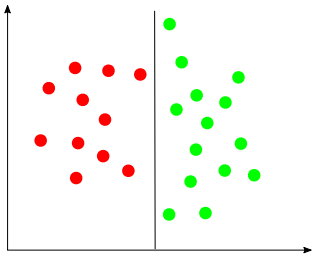

Figure 2.4: A different machine learning model for this dataset.

The process of *automatically* finding better and better representations of our input data is the main aim of machine learning. In other words, applying tools of machine learning to the above problem would automatically find any suitable data transformations of the given training data that would yield a highly accurate classification solution.

Before we discuss the concept of deep learning, we give one last general taxonomy of machine learning approaches, of which there are three:

1. *Supervised learning:* Provide examples of input-output pairs (so-called labelled data), and let the machine learning system learn a function mapping input to output.
2. *Unsupervised learning:* Only the raw data itself is provided and the machine learning system attempts to extract useful features by itself.
3. *Reinforcement learning:* An agent (e.g. some software or a robot) learns to take actions in an environment to maximize a predefined reward.

Deep learning (as well as standard machine learning) is dominated by supervised learning strategies (in particular for visual tasks). The binary classification problem introduced above is a supervised learning problem as well, since we are provided with the labels (red or green) of the data points. Supervised learning has yielded the most substantial breakthroughs in modern machine learning, however, it is a rather costly endeavour as it requires the availability of a labelled dataset. These are not easy (free of cost) to obtain. For example, a human would have to label images of animals, which then would form the training dataset for a machine learning model. Unsupervised learning is much more cost-efficient in this regard, as it would cost nothing to batch-download a huge set of available images off the internet. However, extracting information from unlabelled, raw data is not a trivial task. Still, in this first project we will train an unsupervised computer vision model.

So what is *deep learning*? Deep learning is devoted to training deep artificial neural networks. The 'deep' in deep learning hence refers to having neural networks with multiple layers, rather than a 'deep' such as 'deep meaning'!

Neural networks are actually not a new concept, with the earliest examples dating back to the 1940s. Their popularity has followed a wave-like pattern for several decades until their eventual

breakthrough came in 2012, when they have outperformed all other machine learning based models in the ImageNet competition. The reason for this rather late breakthrough is due to deep neural networks requiring vast amounts of data as well as dedicated hardware (in particular GPUs) for their training that simply were not available up until rather recently. In less than a decade since deep neural networks have been responsible for virtually all breakthroughs in artificial intelligence, including self-driving cars, superhuman achievements in many board and video games, intelligent assistants, and the generation, classification and regression of texts, images, audio and videos.

Most fundamentally a neural network is a stack of multiple layers each of which consists of one or more units (often called neurons) that perform simple mathematical operations to its input values. Fig. 2.5 shows a simple four-layer neural network with two inputs units and two output units. It also has two so-called hidden layers (in that they are neither input nor output layers), with 2 and 3 units, respectively. We call a network like this fully connected, since every unit of the subsequent layer is connected to every unit of the preceding layer.

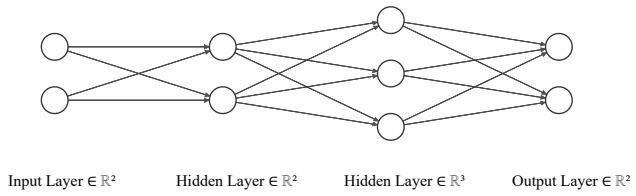

Figure 2.5: A four layer fully connected neural network

Each unit in a neural network can take on one real value. That is, in the network shown in Fig. 2.5 the input layer can accept a two-dimensional vector $\mathbf{x} \in \mathbb{R}^2$. This vector is then passed on to another two-dimensional vector $\mathbf{x}' \in \mathbb{R}^2$ in the first hidden layer, then to a three-dimensional vector $\mathbf{x}'' \in \mathbb{R}^3$ in the second hidden layer before finally being passed on to the output vector $\mathbf{y} \in \mathbb{R}^2$ which is again two-dimensional.

How exactly is one vector passed on to another vector? Fundamentally, each layer in a neural network carries out an elementary mathematical operation, which for the input to the first hidden layer is:

$$\mathbf{x}' = \phi(W_1\mathbf{x} + \mathbf{b}_1),$$

where $W_1 \in \mathbb{R}^{2\times 2}$ is a 2×2 matrix, the so-called weight matrix, and $\mathbf{b}_1 \in \mathbb{R}^2$ is a two-dimensional vector, the so-called bias vector. The function $\phi(\mathbf{z})$ is a nonlinear function, called the *activation function.*

Similarly, the transformation from the first hidden layer to the second hidden layer is

$$\mathbf{x}'' = \phi(W_2\mathbf{x}' + \mathbf{b}_2),$$

were $W_2 \in \mathbb{R}^{2\times 3}$ is now a 2×3 weight matrix, and $\mathbf{b}_2 \in \mathbb{R}^3$ is now a three-dimensional bias vector. Lastly, the transformation from the second hidden layer to the output layer is

$$\mathbf{y} = \phi(W_3\mathbf{x}'' + \mathbf{b}_3),$$

with $W_3 \in \mathbb{R}^{3\times 2}$ being a 3×2 weight matrix, and $\mathbf{b}_3 \in \mathbb{R}^2$ being a two-dimensional bias vector.

Composing all these elementary transformation we see that the mapping from the input vector $\mathbf{x}$ to the output vector $\mathbf{y}$ is given by the nested function:

$$\mathbf{y} = \phi(W_3\phi(W_2\phi(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3).$$

A neural network is then trained by finding good values for the weight matrices and the bias vectors. In the neural network presented above, there is a total of

$$(2 \times 2 + 2) + (2 \times 3 + 3) + (3 \times 2 + 2) = 6 + 9 + 8 = 23$$

parameters that have to be learned. Modern neural networks such as ChatGPT often have billions of parameters, which makes training them a challenging and extremely costly endeavour!

How exactly are neural networks trained then? The simplest setting is the *supervised learning problem*. Here, labelled data in the form of input–output pairs $(\mathbf{x}, \mathbf{y}')$ are provided. The goal is then to choose the weight matrices and bias vectors so that the true labels $\mathbf{y}'$ are as accurately as possible reproduced by the neural network outputs $\mathbf{y}$. In other words, we want that $L = \|\mathbf{y} - \mathbf{y}'\| \to 0$, where L is referred to as a *loss function*. This is an optimization problem that can be solved algorithmically. The method of choice for neural networks is called *gradient descent*, and will be the topic of our project here.

The key ingredient in training a neural network is to define a *loss function* that is a measure of how closely the network fits the given labelled data. There are many different loss functions in use depending on the type of problem to be solved, such as regression or classification tasks. For the purpose of this project, we solely focus on regression problems, which aims to make sure that the predicted quantity $\mathbf{y}$ of the neural network is as close as possible to the true quantity $\mathbf{y}'$ for any given input $\mathbf{x}$ to the network. The most classical example for a regression task is linear regression, which models the relationship between $\mathbf{x}$ and $\mathbf{y}$ as a linear function $\mathbf{y} = W\mathbf{x} + \mathbf{b}$ where the goal is to find the best 'slope' W and the 'intercept' $\mathbf{b}$. It can be easily seen that linear regression can be interpreted as a neural network using just an input and an output layer, and applying a linear activation function.

For regression problems, the suitable loss function is the mean-squared error, or MSE, which we have already used in the first project. For the neural network we have introduced earlier, this translates to minimizing the function

$$L(W_1, W_2, W_3, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{y}_i' - \mathbf{y}_i||^2,$$

where N is the number of training samples of the form $\{(\mathbf{x}_i, \mathbf{y}_i')\}|_{i=1,\dots,N}$. Note that the loss function is a function of the unknown weight matrices and bias vectors, since these are the only unknowns of the given problem. The goal is then to find these weights and biases so that $L \to 0$. This is referred to as training the neural network.

In the following, to simplify the notation we collect all weights and biases in a single network parameter vector $\mathbf{p} = (p_1, \dots, p_d) \in \mathbb{R}^d$, where d equals the number of weights and biases of the network (e.g. $d = 23$ in our network above). The loss function L then defines a mapping $L \colon \mathbb{R}^d \to \mathbb{R}$, and we find the minimum of this loss function using what is referred to as *gradient descent*. We introduce the main idea of gradient descent now.[1]

Assume the current values for the network parameters are $\mathbf{p}$ and we want to update this vector by adding a small increment $\Delta\mathbf{p}$ so that the loss function $L(\mathbf{p} + \Delta\mathbf{p}) < L(\mathbf{p})$. If $\Delta\mathbf{p}$ is small then we can use the linear Taylor series approximation

$$L(\mathbf{p} + \Delta\mathbf{p}) \approx L(\mathbf{p}) + \sum_{j=1}^{d} \frac{\partial L}{\partial p_j} \Delta p_j =: L(\mathbf{p}) + \nabla L(\mathbf{p})^{\mathrm{T}} \Delta\mathbf{p},$$

[1] Note that for the sake of completeness we formulate this algorithm for a general multivariate function. If you have not seen such functions yet, simply set $d = 1$ in the following, which will reduce the multivariate loss function to a univariate real-valued function, which notably reduces the gradient of that function to simply the first derivative.

where we have introduced the gradient vector

$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial p_1} \\ \frac{\partial L}{\partial p_2} \\ \vdots \\ \frac{\partial L}{\partial p_d} \end{pmatrix}.$$

The gradient vector is thus simply the column vector of all first order partial derivatives of a given multivariate function.

We can reduce the loss function if we choose $\Delta\mathbf{p}$ to make $\nabla L(\mathbf{p})^{\mathrm{T}}\Delta\mathbf{p}$ as negative as possible. Using the Cauchy–Schwarz inequality we have for $\mathbf{f}, \mathbf{g} \in \mathbb{R}^d$ that $|\mathbf{f}^{\mathrm{T}}\mathbf{g}| \leqslant ||\mathbf{f}||\,||\mathbf{g}||$. Thus, the most negative that $\mathbf{f}^{\mathrm{T}}\mathbf{g}$ can be is $-||\mathbf{f}||\,||\mathbf{g}||$ which happens if $\mathbf{f} = -\mathbf{g}$.

The above implies that we should choose $\Delta\mathbf{p}$ to lie in the direction of $-\nabla L(\mathbf{p})$. Note that this only gives us the direction in which to change the parameter vector $\mathbf{p}$, but it does not yet give us the step size in that direction which we have to take. Thus, we multiply this direction with a small parameter $\eta \in \mathbb{R}$, which in machine learning is called the *learning rate*. This yields the update rule

$$\mathbf{p} \leftarrow \mathbf{p} - \eta\nabla L(\mathbf{p}). \tag{2.1}$$

Note that the learning rate has to be small as otherwise the assumption that we can ignore quadratic terms in our above Taylor series would be violated!

The above formula gives the following heuristic update rule: Assume that initially the weights and biases of the neural network are initialized to some values $\mathbf{p}^{(0)}$. We can then iteratively update these values using the update rule

$$\mathbf{p}^{(n+1)} = \mathbf{p}^{(n)} - \eta\nabla L(\mathbf{p}^{(n)}), \quad n = 0, \ldots$$

This formula requires knowledge of the gradient of the loss function, which fortunately can be computed easily using the chain rule for any given neural network. In practice, one then monitors the value of the loss function as a function of the iteration step n and terminates this iteration when the difference between two consecutive steps becomes smaller than a user-defined threshold.

The update rule (2.1) is referred to as *gradient descent*. While a straightforward way for minimizing the loss function, it can be rather slow to converge in practice, and thus many different variation of this main idea for training a neural network were formulated in the past few years.

Momentum optimization. One key addition to gradient descent consists of *momentum*. The idea is motivated from physics where letting a ball roll down a slop will increase momentum, thus being able to progress faster downhill than using no momentum (as in gradient descent, which just takes regular small steps downhill).

In standard gradient descent, the weight vectors are updated using formula (2.1). Note that none of the previous gradients factor in to this weight update rule.

In momentum optimization the previous gradients do factor into the gradient update. Specifically, at each iteration a momentum optimizer subtracts the local gradient from the momentum vector (scaled by the learning rate) and updates the weights by adding this momentum vector. To prevent the momentum from growing too large, also a friction mechanism is introduced (just

as a ball rolling down a hill is being slowed down by friction). The full momentum optimization algorithm is

$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla L(\mathbf{p})$$
$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{m},$$

where β is a hyperparameter controlling the fiction (0 for high friction, reducing the algorithm to standard gradient descent, and 1 for no friction)

Let us consider the following simplified example, where we assume the gradients of the loss function to be constant, to illustrate how momentum optimization can improve upon standard gradient descent. For constant gradients, the maximum size of the weight updates (i.e. the terminal velocity) is equal to the gradient multiplied by the learning rate multiplied by $1/(1-\beta)$ (geometric series). Thus if $\beta = 0.9$ then the weight updates are 10 times the gradient times the learning rates, making momentum optimizers going 10 times faster than standard gradient descent. This allows momentum optimizers to escape plateaus (or local minima) faster than gradient descent, thus speeding up learning. As a downside, due to the momentum, the optimizer may overshoot the minimum by a bit and then oscillate a few times before stabilizing at the minimum.

Nesterov accelerated momentum. There is a small modification to the above described momentum optimization, which is called *Nesterov accelerated momentum*. Rather than evaluating the gradient at the position $\mathbf{p}$, one evaluates it ahead in the direction of the momentum at $\mathbf{p} + \beta\mathbf{m}$. The full Nesterov momentum optimization algorithm is

$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla L(\mathbf{p} + \beta\mathbf{m}),$$
$$\mathbf{p} \leftarrow \mathbf{p} + \mathbf{m},$$

The idea is that the momentum vector will point in the right direction so it may be more accurate to use the gradient evaluated towards the new position, rather than at the old position. These small improvements over time can add up so that the Nesterov accelerated gradient algorithm can be significantly faster than the standard momentum algorithm.

AdaGrad. AdaGrad (adaptive gradient algorithm), aims at speeding up convergence by correcting the direction of the gradient to point more towards the global optimum, ideally reducing the oscillatory behaviour of the vanilla momentum optimizer.

AdaGrad does this by scaling down the gradient vector of the steepest dimension,

$$\mathbf{s} \leftarrow \mathbf{s} + \nabla L(\mathbf{p}) \otimes \nabla L(\mathbf{p}),$$
$$\mathbf{p} \leftarrow \mathbf{p} - \eta\nabla L(\mathbf{p}) \oslash \sqrt{\mathbf{s} + \varepsilon}.$$

The first step accumulates the squared gradient components ($\otimes$ is element-wise multiplication) into a vector, which will yield larger and larger components s_i at each iteration if the loss function is steep along the i-th direction. The second step is similar to gradient descent, except now the gradient vector is scaled down by a factor of $\sqrt{\mathbf{s} + \varepsilon}$ ($\oslash$ is element-wise division), where $\varepsilon \sim 10^{-10}$ is a regularization parameter to avoid division by zero. This scaling operation effectively decays the learning rate but does this faster for steeper dimensions than for those with gentler slopes. AdaGrad performs well for simple quadratic problems but often stops too early when training neural networks, as the learning rate gets scaled down too much. As such, it cannot be used for training neural networks.

RMSprop. The key idea of AdaGrad is to effectively introduce an *adaptive learning rate*. The gradient descent algorithm aims to find the most sensitive directions of the loss function and balances the learning rate to ideally avoid overshooting. AdaGrad unfortunately slows down the learning rate too much, so that for neural networks a global minimum is hardly ever reached. The RMSProp (Root mean square propagation) optimizer was introduced by Geoff Hinton and Tijmen Tieleman in 2012 to fix this shortcoming. The key difference is to accumulate the gradients only from the most recent iterations, as opposed to since beginning of training as in AdaGrad.

This is done using exponential decay in the first step of AdaGrad,

$$\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla L(\mathbf{p}) \otimes \nabla L(\mathbf{p}),$$
$$\mathbf{p} \leftarrow \mathbf{p} - \eta\nabla L(\mathbf{p}) \oslash \sqrt{\mathbf{s}+\varepsilon},$$

where the *decay rate* β is typically set to $\beta = 0.9$.

While introducing yet another hyperparameter, the value of $\beta = 0.9$ typically works reasonably well in practice. RMSProp almost always performs much better than AdaGrad and is still one of the most used optimizers in practice.

Adam. Adam (adaptive moment estimation) combines the ideas of momentum optimization and RMSProp. Adam keeps track of an exponentially decaying average of past gradients as well as of an exponentially decaying average of past squared gradients.

Indexing by t the training iteration (beginning at 0), the Adam update is

$$\begin{aligned}
\text{Step } 0: &\quad t \leftarrow t+1\\
\text{Step } 1: &\quad \mathbf{m} \leftarrow \beta_1\mathbf{m} + (1-\beta_1)\nabla L(\mathbf{p})\\
\text{Step } 2: &\quad \mathbf{s} \leftarrow \beta_2\mathbf{s} + (1-\beta_2)\nabla L(\mathbf{p}) \otimes \nabla L(\mathbf{p})\\
\text{Step } 3/4: &\quad \hat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-\beta_1^t}, \quad \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\beta_2^t}\\
\text{Step } 5: &\quad \mathbf{p} \leftarrow \mathbf{p} - \eta\hat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}}+\varepsilon}.
\end{aligned}$$

Steps 1, 2 and 5 are taken from momentum optimization and RMSProp, computing an exponentially decaying average of momentum and gradients, and updating the weights accordingly. Steps 3 and 4 try to boost $\mathbf{m}$ and $\mathbf{s}$ at the beginning of training since these vectors are initialized to $\mathbf{0}$ and thus biased toward $\mathbf{0}$ initially.

The momentum decay hyperparameter β_1 is usually set to 0.9, and the scaling decay hyperparameter β_2 is usually set to 0.999. The learning rate η typically does not have to be tuned since Adam is an adaptive learning rate algorithm, making it relatively straightforward to use with the default $\eta = 0.001$. Adam is by far the most popular optimizer nowadays. Variants also exist that implement Adam with Nesterov momentum (Nadam) but we will not use them here.

2.2 Project

2.2.1 A study on neural network optimizers in 1D.

Consider the loss function

$$L(p) = \sin 2p + a \sin 4p,$$

where $a \in \mathbb{R}$ is a real constant. We thus consider an optimization problem where L is interpreted as a loss function depending on a single unknown parameter p. For this case, the gradient of the

function L with respect to p is simply the first-order derivative, i.e. $\nabla L = L' = \frac{\mathrm{d}L}{\mathrm{d}p}$. Complete the following tasks:

1. Implement all optimizers we have seen above using `Python`, i.e. gradient descent, the vanilla momentum optimizer, the Nesterov accelerated momentum optimizer, AdaGrad, RMSProp and Adam.
2. Test these optimizers for the function $L(p)$ with $a = 0.499$ with learning rates $\eta \in \{0.1, 0.01, 0.001\}$ using the initial guess $p_0 = 0.75$. How many steps does it take each optimizer to find the minimum of this function? Use an absolute (target) error of 10^{-13} as a stopping criterion, i.e. stop the minimization procedure when $|L(p^{(n+1)}) - L(p^{(n)})| \leqslant 10^{-13}$ for two consecutive steps n and $n+1$.
3. Repeat the above steps for $a = 0.501$.
4. Interpret your results!

2.2.2 A study on neural network optimizers in 2D.

To visualize the path an optimizer takes along a loss surface we consider a two-dimensional optimization problem here. There are many complicated functions that have been studied extensively in the field of optimization owing to their properties of having many local minimal, flat regions, or valleys, which make the optimization procedure potentially challenging. A list of such functions of the form $L = L(p_1, p_2)$ can be found here:

https://www.sfu.ca/~ssurjano/optimization.html

Complete the following tasks:

1. Visualize some of the functions from the above link, in particular the *Six hump camel function* and the *Michalewicz function*, but feel free to choose at least two more suitable functions as well for your report.
2. Use the optimizers from the previous question, i.e. gradient descent, the vanilla momentum optimizer, the Nesterov accelerated momentum optimizer, AdaGrad, RMSProp and Adam to find a minimum of these functions. Then visualize the trajectory taken by the optimizer from a starting point $(p_1^{\text{start}}, p_2^{\text{start}})$ to the minimum $(p_1^{\text{min}}, p_2^{\text{min}})$ of the considered function $L = L(p_1, p_2)$. You could visualize this both in 2D (i.e. projected to the plane spanned by p_1 and p_2) or in 3D (i.e. visualizing the path along the loss surface itself taken by the optimizer). An example of how this could look is given in Figure 2.6.
3. Indicate which optimizers fail to reach a minimum and which get stuck in a local rather than the global minimum.
4. In addition, keep track of the total number of steps each optimizer required to find a minimum of the considered function. Use the same absolute target error of the previous question.

Hint: Note that some of the optimizers will not succeed in finding an appropriate minimum. As such, it may be required to limit the total number of optimization steps an optimizer is allowed to take, after which the optimization procedure will be terminated even if the target error was not reached. Set this maximum number of steps to a large number, such as 100,000 or 1,000,000.

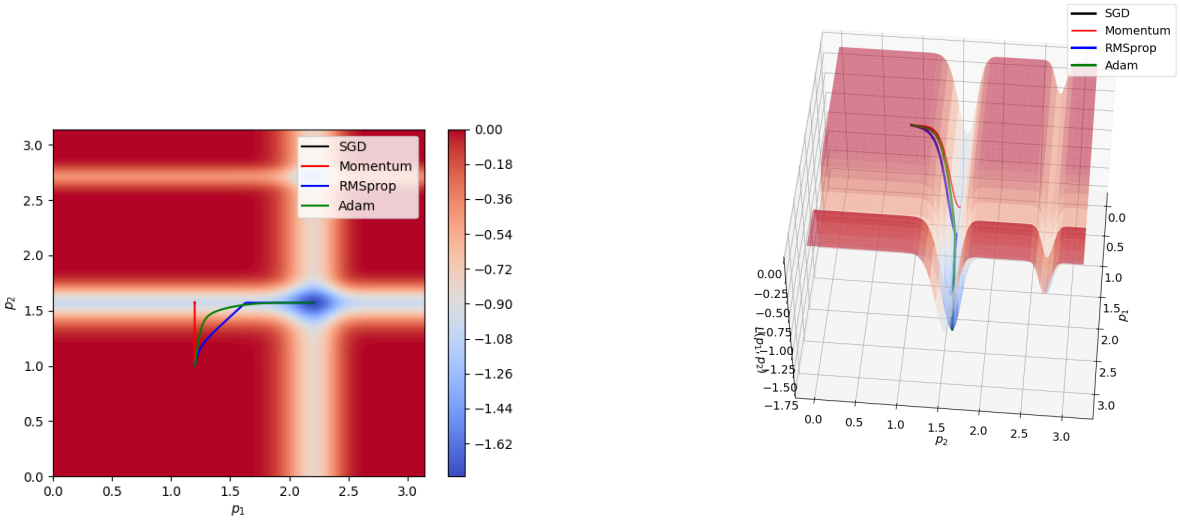

Figure 2.6: Sample results for a two-dimensional optimization example. *Left:* Two-dimensional projection of loss surface and optimization paths. *Right:* Three-dimensional loss surface and optimization paths.

2.2.3 Training a neural network

Having investigated how to minimize a function in one and two variables we can now generalize our optimizers to train an actual neural network. As introduced in Section 2.1, a neural network is simply a composite function, and as such merely a more complicated function than considered in the previous two tasks. The goal of this final task for project 2 is to train your own neural network using the optimizers you have implemented so far.

We consider a neural network in the supervised learning setting. In particular, we want to verify that a neural network can indeed approximate a given function, which is guaranteed by the *Universal Approximation Theorem* [1]. We consider the function

$$f(x) = (1 - x)\sin 2x + (1 - x)^2 \sin 10x, \tag{2.2}$$

and aim to approximate this function with a neural network over the domain $x \in [0, 1]$.

Complete the following tasks:

1. Generate your training dataset. Sample the given function at n regularly spaced grid points over the domain $[0, 1]$. Experiment with $n \in \{32, 64, 128\}$.
2. Train a neural network with h hidden layers using u units per hidden layer, employing the hyperbolic tangent activation function. Experiment with $h \in \{2, 4, 6\}$ and $u \in \{16, 32, 64\}$. The loss function to be used is the mean squared error, and the learning rate should be varied over a reasonable range, such as $\eta \in [10^{-4}, 1.0]$. Use all the optimizers we have introduced in this module and train each neural network for 10,000 optimization steps. You can also experiment with more (or fewer) optimization steps until the loss converges to its minimum but this may require, depending on the learning rate you use, more than 1 million optimization steps, and thus might take some time!
3. Evaluate the performance of your trained neural networks by plotting the reconstructed function along with the mean squared error obtained. Also plot the time series of the loss

to assess the performance of each optimizer. An example of how this could look like for some of the optimizers to be considered is given in Figure 2.7.

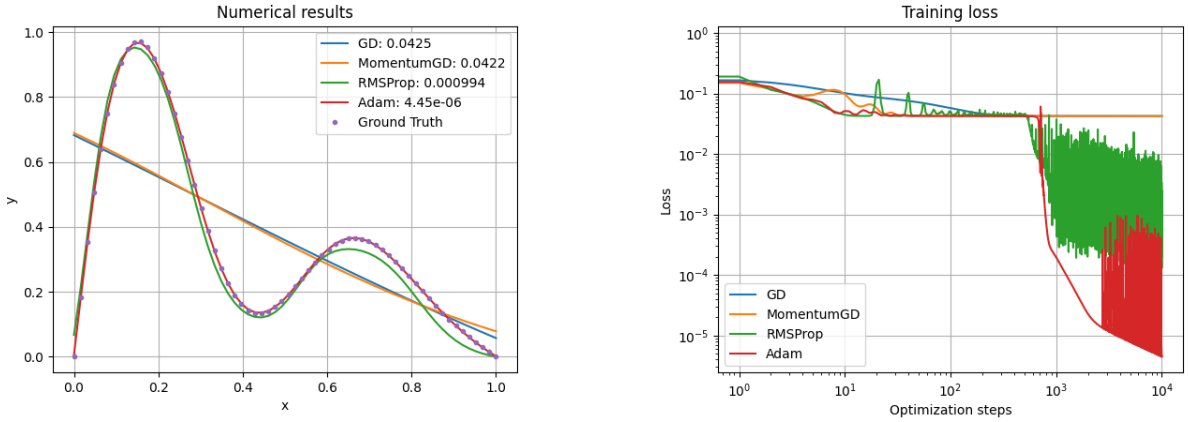

Figure 2.7: Sample results for a trained neural network to approximate function (2.2). *Left:* Reconstruction of that function using a neural network with two hidden layers with 32 units per layer using a learning rate of $\eta = 0.01$. *Right:* Time series of loss.

Remark 3. Sample code for this task is provided on Brightspace. This code trains a neural network in `JAX` using standard gradient descent. For the purpose of this project the amount of `JAX` required is rather minimal, but feel free to learn more about it as this is one of the main packages being used today to train neural networks, especially in a research setting.

2.3 Why this project is relevant

Optimization is at the core of deep learning. Every single deep learning model being used in practice today has been trained with one of the optimizers used in this task. It is one often forgotten component of training deep learning based models to assess the impact that the optimization method has on the performance of the trained model. As this project has demonstrated, the minima found of the same loss function using different optimization algorithms may be different, which would in practice translate to potentially starkly different performance in the resulting trained neural networks. In other words, using the exact same problem set-up (same dataset, and same architecture for the neural network) but a different optimization algorithm may lead to distinctly performing models.

Optimization is not only a core element of deep learning, but also of the mathematical sciences in general. Optimization problems have to be solved in fields as diverse as astronomy, biology, engineering, finance, geophysics, and meteorology. As such, learning about optimization provides you an important tool for your toolbox as an (applied) mathematician.

2.4 Further reading

If the field of deep learning interests you (and it should, as it is one of the major breakthroughs of science in the last 100 years), there are many books available on this subject that are accessible to readers with background knowledge in calculus and linear algebra. In other words, this field

should be accessible to any second year math student! Two resources to list here are [3] (more theoretical) and [2] (more focused on practical implementation).

Bibliography

[1] Cybenko G., Approximation by superpositions of a sigmoidal function, *Math. Control Signals Syst.* **2** (1989), 303–314.

[2] Géron A., *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, 2019.

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

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

[5] Krantz S.G., *A primer of mathematical writing*, vol. 243, American Mathematical Soc., 2017.