Project 2 on Machine Learning, deadline November 4 (Midnight)

Data Analysis and Machine Learning FYS-STK3155/FYS4155

Department of Physics, University of Oslo, Norway

Oct 8, 2024

Classification and Regression, from linear and logistic regression to neural networks

The main aim of this project is to study both classification and regression problems by developing our own feed-forward neural network (FFNN) code. We can reuse the regression algorithms studied in project 1. We will also include logistic regression for classification problems and write our own FFNN code for studying both regression and classification problems. The codes developed in project 1, including bootstrap **and/or** cross-validation as well as the computation of the mean-squared error and/or the $R2$ or the accuracy score (classification problems) functions can also be utilized in the present analysis.

The data sets that we propose here are (the default sets)

- Regression (fitting a continuous function). In this part you will need to bring back your results from project 1 and compare these with what you get from your Neural Network code to be developed here. The data sets could be
  1. A simple one-dimensional function or the Franke function or the terrain data from project 1, or data sets your propose. It could be a simpler function than the Franke function. We recommend testing a simpler function (see below). But if you wish to try more complex function, feel free to do so.

- Classification. Here you will also need to develop a Logistic regression code that you will use to compare with the Neural Network code. The data set we propose are the so-called Wisconsin Breat Cancer Data data set of images representing various features of tumors. A longer explanation with links to the scientific literature can be found at the Machine Learning repository of the University of California at Irvine. Feel free to consult this site and the pertinent literature.

© 1999-2024, "Data Analysis and Machine Learning FYS-STK3155/FYS4155":"http://www.uio.no/studier/emner/matnat/fys/FYS3155/index-eng.html". Released under CC Attribution-NonCommercial 4.0 license

You can find more information about this at the Scikit-Learn site or at the University of California at Irvine.

However, if you would like to study other data sets, feel free to propose other sets. What we list here are mere suggestions from our side. If you opt for another data set, consider using a set which has been studied in the scientific literature. This makes it easier for you to compare and analyze your results. Comparing with existing results from the scientific literature is also an essential element of the scientific discussion. The University of California at Irvine with its Machine Learning repository at `https://archive.ics.uci.edu/ml/index.php` is an excellent site to look up for examples and inspiration. Kaggle.com is an equally interesting site. Feel free to explore these sites.

We will start with a regression problem and we will reuse our codes from project 1 starting with writing our own Stochastic Gradient Descent (SGD) code.

Part a): Write your own Stochastic Gradient Descent code, first step. In order to get started, we will now replace in our standard ordinary least squares (OLS) and Ridge regression codes (from project 1) the matrix inversion algorithm with our own gradient descent (GD) and SGD codes. You can use the Franke function or the terrain data from project 1. **However, we recommend using a simpler function like** $f(x) = a_0 + a_1x + a_2x^2$ or higher-order one-dimensional polynomials. You can obviously test your final codes against for example the Franke function.

The exercise set for week 41 should help in solving this part of the project.

You should include in your analysis of the GD and SGD codes the following elements

1. A plain gradient descent with a fixed learning rate (you will need to tune it) using the analytical expression for the gradient.

2. Add momentum to the plain GD code and compare convergence with a fixed learning rate (you may need to tune the learning rate). Keep using the analytical expression for the gradient.

3. Repeat these steps for stochastic gradient descent with mini batches and a given number of epochs. Use a tunable learning rate as discussed in the lectures from weeks 39 and 40. Discuss the results as functions of the various parameters (size of batches, number of epochs etc). Use the analytical gradient.

4. Implement the Adagrad method in order to tune the learning rate. Do this with and without momentum for plain gradient descent and SGD.

5. Add RMSprop and Adam to your library of methods for tuning the learning rate.

The lecture notes from weeks 39 and 40 contain more details and code examples. Feel free to use these examples.

1. Replace thereafter your analytical gradient with either **Autograd** or **JAX**

Feel free to use codes on these methods from the lecture notes from week 39 and week 40.

In summary, you should perform an analysis of the results for OLS and Ridge regression as function of the chosen learning rates, the number of mini-batches and epochs as well as algorithm for scaling the learning rate. You can also compare your own results with those that can be obtained using for example **Scikit-Learn**'s various SGD options. Discuss your results. For Ridge regression you need now to study the results as functions of the hyper-parameter λ and the learning rate η. Discuss your results.

You will need your SGD code for the setup of the Neural Network and Logistic Regression codes. You will find the Python Seaborn package useful when plotting the results as function of the learning rate η and the hyper-parameter λ when you use Ridge regression. Since you will use different gradient descent methods, you can also add Lasse regression. This is however optional. How to code Lasso regression is discussed in the lecture notes from week 40.

We recommend reading chapter 8 on optimization from the textbook of Goodfellow, Bengio and Courville at `https://www.deeplearningbook.org/`. This chapter contains many useful insights and discussions on the optimization part of machine learning.

Part b): Writing your own Neural Network code. Your aim now, and this is the central part of this project, is to write your own Feed Forward Neural Network code implementing the back propagation algorithm discussed in the lecture slides from week 41 and week 42.

We will focus on a regression problem first and study either the simple second-order polynomial from part a) or the Franke function or terrain data (or both or other data sets) from project 1.

Discuss again your choice of cost function.

Write an FFNN code for regression with a flexible number of hidden layers and nodes using the Sigmoid function as activation function for the hidden layers. Initialize the weights using a normal distribution. How would you initialize the biases? And which activation function would you select for the final output layer?

Train your network and compare the results with those from your OLS and Ridge Regression codes from project 1 if you use the Franke function or the terrain data. You should test your results against a similar code using **Scikit-Learn** (see the examples in the above lecture notes from weeks 41 and 42) or **tensorflow/keras** or **Pytorch** (for Pytorch, see Raschka et al.'s text chapters 12 and 13).

Comment your results and give a critical discussion of the results obtained with the Linear Regression code and your own Neural Network code. Make an analysis of the regularization parameters and the learning rates employed to find the optimal MSE and $R2$ scores.

A useful reference on the back progagation algorithm is Nielsen's book at `http://neuralnetworksanddeeplearning.com/`. It is an excellent read.

Part c): Testing different activation functions. You should now also test different activation functions for the hidden layers. Try out the Sigmoid, the RELU and the Leaky RELU functions and discuss your results. You may also study the way you initialize your weights and biases.

Part d): Classification analysis using neural networks. With a well-written code it should now be easy to change the activation function for the output layer.

Here we will change the cost function for our neural network code developed in parts b) and c) in order to perform a classification analysis.

We will here study the Wisconsin Breast Cancer data set. This is a typical binary classification problem with just one single output, either True or Fale, 0 or 1 etc. You find more information about this at the Scikit-Learn site or at the University of California at Irvine.

To measure the performance of our classification problem we use the so-called *accuracy* score. The accuracy is as you would expect just the number of correctly guessed targets t_i divided by the total number of targets, that is

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n},$$

where I is the indicator function, 1 if $t_i = y_i$ and 0 otherwise if we have a binary classification problem. Here t_i represents the target and y_i the outputs of your FFNN code and n is simply the number of targets t_i.

Discuss your results and give a critical analysis of the various parameters, including hyper-parameters like the learning rates and the regularization parameter λ (as you did in Ridge Regression), various activation functions, number of hidden layers and nodes and activation functions.

As stated in the introduction, it can also be useful to study other datasets.

Again, we strongly recommend that you compare your own neural Network code for classification and pertinent results against a similar code using **Scikit-Learn** or **tensorflow/keras** or **pytorch**.

Part e): Write your Logistic Regression code, final step. Finally, we want to compare the FFNN code we have developed with Logistic regression, that is we wish to compare our neural network classification results with the results we can obtain with another method.

Define your cost function and the design matrix before you start writing your code. Write thereafter a Logistic regression code using your SGD algorithm. You can also use standard gradient descent in this case, with a learning rate as hyper-parameter. Study the results as functions of the chosen learning rates. Add also an l_2 regularization parameter λ. Compare your results with those from your FFNN code as well as those obtained using **Scikit-Learn**'s logistic regression functionality.

The weblink here https://medium.com/ai-in-plain-english/comparison-between-logistic-regress logistic regression and FFNN using the so-called MNIST data set. You may find several useful hints and ideas from this article.

Part f) Critical evaluation of the various algorithms. After all these glorious calculations, you should now summarize the various algorithms and come with a critical evaluation of their pros and cons. Which algorithm works best for the regression case and which is best for the classification case. These codes can also be part of your final project 3, but now applied to other data sets.

Background literature

1. The text of Michael Nielsen is highly recommended, see Nielsen's book at http://neuralnetworksanddeeplearning.com/. It is an excellent read.
2. Goodfellow, Bengio and Courville, Deep Learning at https://www.deeplearningbook.org/. Here we recommend chapters 6, 7 and 8
3. Raschka et al. at https://sebastianraschka.com/blog/2022/ml-pytorch-book.html. Here we recommend chapters 11, 12 and 13.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.

- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Use Canvas to hand in your projects, log in at `https://www.uio.no/english/services/it/education/canvas/` with your normal UiO username and password.
- Upload **only** the report file or the link to your GitHub/GitLab or similar typo of repos! For the source code file(s) you have developed please provide us with your link to your GitHub/GitLab or similar domain. The report file should include all of your discussions and a list of the codes you have developed. Do not include library files which are available at the course homepage, unless you have made specific changes to them.
- In your GitHub/GitLab or similar repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.

Finally, we encourage you to collaborate. Optimal working groups consist of 2-3 students. You can then hand in a common report.