Project2

October 21, 2025

1 Project 2 on Machine Learning, deadline November 10 (Midnight)

Data Analysis and Machine Learning FYS-STK3155/FYS4155, University of Oslo, Norway

Date: **October 14, 2025**

1.1 Deliverables

First, join a group in canvas with your group partners. Pick an avaliable group for Project 2 in the **People** page.

In canvas, deliver as a group and include:

- A PDF of your report which follows the guidelines covered below and in the week 39 exercises. Additional requirements include:
  - It should be around 5000 words, use the word counter in Overleaf for this. This often corresponds to 10-12 pages. References and appendices are excluded from the word count
  - It should include around 10-15 figures. You can include more figures in appendices and/or as supplemental material in your repository.
- A comment linking to your github repository (or folder in one of your github repositories) for this project. The repository must include

A PDF file of the report * A folder named Code, where you put python files for your functions and notebooks for reproducing your results. Remember to use a seed for generating random data and for train-test splits when generating final results.

- A README file with the name of the group members
- a short description of the project
- a description of how to install the required packages to run your code from a requirements.txt file or similar (such as a plain text description) names and descriptions of the various notebooks in the Code folder and the results they produce

1.1.1 Preamble: Note on writing reports, using reference material, AI and other tools

We want you to answer the three different projects by handing in reports written like a standard scientific/technical report. The links at

https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects contain more information. There you can find examples of previous reports, the projects themselves, how we grade reports etc. How to write reports will also be discussed during the various lab sessions. Please do ask us if you are in doubt.

When using codes and material from other sources, you should refer to these in the bibliography of your report, indicating wherefrom you for example got the code, whether this is from the lecture notes, softwares like Scikit-Learn, TensorFlow, PyTorch or other sources. These sources should always be cited correctly. How to cite some of the libraries is often indicated from their corresponding GitHub sites or websites, see for example how to cite Scikit-Learn at https://scikit-learn.org/dev/about.html.

We enocurage you to use tools like ChatGPT or similar in writing the report. If you use for example ChatGPT, please do cite it properly and include (if possible) your questions and answers as an addition to the report. This can be uploaded to for example your website, GitHub/GitLab or similar as supplemental material.

If you would like to study other data sets, feel free to propose other sets. What we have proposed 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.

1.2 Classification and Regression, writing our own neural network code

The main aim of this project is to study both classification and regression problems by developing our own feed-forward neural network (FFNN) code. The exercises from week 41 and 42 (see https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/exercisesweek41.html and https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/exercisesweek42.html) as well as the lecture material from the same weeks (see https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week41.html and https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week42.html) should contain enough information for you to get started with writing your own code.

We will also reuse our codes on gradient descent methods from project 1.

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
  - The simple one-dimensional function Runge function from project 1, that is $f(x) = \frac{1}{1+25x^2}$. We recommend using a simpler function when developing your neural network code for regression problems. Feel however free to discuss and study other functions, such as the two-dimensional Runge function $f(x,y) = \left[(10x-5)^2 + (10y-5)^2 + 1\right]^{-1}$, or even more complicated two-dimensional functions (see the supplementary material of https://www.nature.com/articles/s41467-025-61362-4 for an extensive list of

two-dimensional functions).

- Classification.
- We will consider a multiclass classification problem given by the full MNIST data set. The full data set is at https://www.kaggle.com/datasets/hojjatk/mnist-dataset.

We will start with a regression problem and we will reuse our codes on gradient descent methods from project 1.

1.2.1 Part a): Analytical warm-up

When using our gradient machinery from project 1, we will need the expressions for the cost/loss functions and their respective gradients. The functions whose gradients we need are: 1. The mean-squared error (MSE) with and without the L_1 and L_2 norms (regression problems)

2. The binary cross entropy (aka log loss) for binary classification problems with and without L_1 and L_2 norms
3. The multiclass cross entropy cost/loss function (aka Softmax cross entropy or just Softmax loss function)

Set up these three cost/loss functions and their respective derivatives and explain the various terms. In this project you will however only use the MSE and the Softmax cross entropy.

We will test three activation functions for our neural network setup, these are the 1. The Sigmoid (aka **logit**) function,

2. the RELU function and
3. the Leaky RELU function

Set up their expressions and their first derivatives. You may consult the lecture notes (with codes and more) from week 42 at https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week42.html.

1.2.2 Reminder about the gradient machinery from project 1

In the setup of a neural network code you will need your gradient descent codes from project 1. For neural networks we will recommend using stochastic gradient descent with either the RMSprop or the ADAM algorithms for updating the learning rates. But you should feel free to try plain gradient descent as well.

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

You will find the Python Seaborn package useful when plotting the results as function of the learning rate η and the hyper-parameter λ .

1.2.3 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 FFNN code implementing the back propagation algorithm discussed in the lecture slides from

week 41 at https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week41.html and week 42 at https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week42.html.

We will focus on a regression problem first, using the one-dimensional Runge function

$$f(x) = \frac{1}{1+25x^2},$$

from project 1.

Use only the mean-squared error as cost function (no regularization terms) and write an FFNN code for a regression problem with a flexible number of hidden layers and nodes using only 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? And how would you set up your design/feature matrix? Hint: does it have to represent a polynomial approximation as you did in project 1?

Train your network and compare the results with those from your OLS regression code from project 1 using the one-dimensional Runge function. When comparing your neural network code with the OLS results from project 1, use the same data sets which gave you the best MSE score. Moreover, use the polynomial order from project 1 that gave you the best result. Compare these results with your neural network with one and two hidden layers using 50 and 100 hidden nodes, respectively.

Comment your results and give a critical discussion of the results obtained with the OLS code from project 1 and your own neural network code. Make an analysis of the learning rates employed to find the optimal MSE score. Test both stochastic gradient descent with RMSprop and ADAM and plain gradient descent with different learning rates.

You should, as you did in project 1, scale your data.

1.2.4 Part c): Testing against other software libraries

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).

Furthermore, you should also test that your derivatives are correctly calculated using automatic differentiation, using for example the **Autograd** library or the **JAX** library. It is optional to implement these libraries for the present project. In this project they serve as useful tests of our derivatives.

1.2.5 Part d): Testing different activation functions and depths of the neural network

You should also test different activation functions for the hidden layers. Try out the Sigmoid, the RELU and the Leaky RELU functions and discuss your results. Test your results as functions of the number of hidden layers and nodes. Do you see signs of overfitting? It is optional in this project to perform a bias-variance trade-off analysis.

1.2.6 Part e): Testing different norms

Finally, still using the one-dimensional Runge function, add now the hyperparameters λ with the L_2 and L_1 norms. Find the optimal results for the hyperparameters λ and the learning rates η and neural network architecture and compare the L_2 results with Ridge regression from project 1 and the L_1 results with the Lasso calculations of project 1. Use again the same data sets and the best results from project 1 in your comparisons.

1.2.7 Part f): 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), d) and e) in order to perform a classification analysis. The classification problem we will study is the multiclass MNIST problem, see the description of the full data set at https://www.kaggle.com/datasets/hojjatk/mnist-dataset. We will use the Softmax cross entropy function discussed in a). The MNIST data set discussed in the lecture notes from week 42 is a downscaled variant of the full dataset.

Feel free to suggest other data sets. If you find the classic MNIST data set somewhat limited, feel free to try the
MNIST-Fashion data set at for example https://www.kaggle.com/datasets/zalando-research/fashionmnist.

To set up the data set, the following python programs may be useful

[1]:
```
from sklearn.datasets import fetch_openml

# Fetch the MNIST dataset
mnist = fetch_openml('mnist_784', version=1, as_frame=False, parser='auto')

# Extract data (features) and target (labels)
X = mnist.data
y = mnist.target
```

You should consider scaling the data. The Pixel values in MNIST range from 0 to 255. Scaling them to a 0-1 range can improve the performance of some models. That is, you could implement the following scaling

[2]:
```
X = X / 255.0
```

And then perform the standard train-test splitting

[3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

To measure the performance of our classification problem we will 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 hyperparameters like the learning rates and the regularization parameter λ, various activation functions, number of hidden layers and nodes and activation functions.

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**.

If you have time, you can use the functionality of **scikit-learn** and compare your neural network results with those from Logistic regression. This is optional. The weblink here https://medium.com/ai-in-plain-english/comparison-between-logistic-regression-and-neural-networks-in-classifying-digits-dc5e85cd93c3 logistic regression and FFNN using the so-called MNIST data set. You may find several useful hints and ideas from this article. Your neural network code can implement the equivalent of logistic regression by simply setting the number of hidden layers to zero and keeping just the input and the output layers.

If you wish to compare with say Logisti Regression from **scikit-learn**, the following code uses the above data set

[4]:
```
from sklearn.linear_model import LogisticRegression
# Initialize the model
model = LogisticRegression(solver='saga', multi_class='multinomial',␣
 ↪max_iter=1000, random_state=42)
# Train the model
model.fit(X_train, y_train)
from sklearn.metrics import accuracy_score
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
```

1.2.8 Part g) 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.

1.3 Summary of methods to implement and analyze

Required Implementation: 1. Reuse the regression code and results from project 1, these will act as a benchmark for seeing how suited a neural network is for this regression task.

2. Implement a neural network with

- A flexible number of layers
- A flexible number of nodes in each layer
- A changeable activation function in each layer (Sigmoid, ReLU, LeakyReLU, as well as Linear and Softmax)
- A changeable cost function, which will be set to MSE for regression and cross-entropy for multiple-classification
- An optional L1 or L2 norm of the weights and biases in the cost function (only used for computing gradients, not interpretable metrics)

3. Implement the back-propagation algorithm to compute the gradient of your neural network
4. Reuse the implementation of Plain and Stochastic Gradient Descent from Project 1 (and adapt the code to work with the your neural network)

- With no optimization algorithm
- With RMS Prop
- With ADAM

5. Implement scaling and train-test splitting of your data, preferably using sklearn
6. Implement and compute metrics like the MSE and Accuracy

1.3.1 Required Analysis:

1. Briefly show and argue for the advantages and disadvantages of the methods from Project 1.
2. Explore and show the impact of changing the number of layers, nodes per layer, choice of activation function, and inclusion of L1 and L2 norms. Present only the most interesting results from this exploration. 2D Heatmaps will be good for this: Start with finding a well performing set of hyper-parameters, then change two at a time in a range that shows good and bad performance.
3. Show and argue for the advantages and disadvantages of using a neural network for regression on your data
4. Show and argue for the advantages and disadvantages of using a neural network for classification on your data
5. Show and argue for the advantages and disadvantages of the different gradient methods and learning rates when training the neural network

1.3.2 Optional (Note that you should include at least two of these in the report):

1. Implement Logistic Regression as simple classification model case (equivalent to a Neural Network with just the output layer)
2. Compute the gradient of the neural network with autograd, to show that it gives the same result as your hand-written backpropagation.

3. Compare your results with results from using a machine-learning library like pytorch (https://docs.pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)
4. Use a more complex classification dataset instead, like the fashion MNIST (see https://www.kaggle.com/datasets/zalando-research/fashionmnist)
5. Use a more complex regression dataset instead, like the two-dimensional Runge function $f(x,y) = \left[(10x-5)^2 + (10y-5)^2 + 1\right]^{-1}$, or even more complicated two-dimensional functions (see the supplementary material of https://www.nature.com/articles/s41467-025-61362-4 for an extensive list of two-dimensional functions).
6. Compute and interpret a confusion matrix of your best classification model (see https://www.researchgate.net/figure/Confusion-matrix-of-MNIST-and-F-MNIST-embeddings_fig5_349758607)

1.4 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.

1.5 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.

1.6 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.