Mini-Project
Optimization for Machine Learning
Spring 2026

EPFL
School of Computer and Communication Sciences
Nicolas Flammarion & Martin Jaggi
github.com/epfml/OptML_course

Mini-Project
(Submission Deadline: June 12th, 2026)

Introduction

The mini-project focuses on practical implementation. We encourage you to investigate an optimization algorithm for a real-life machine learning application and gain insight into that algorithm. You should provide empirical evidence for the behavior of your chosen optimization algorithm or modification. The studied behavior could be discussed in class or not and could be desirable or undesirable. The optimization algorithms can be anything of your choice. Don't be scared to try variants not seen in class, as the project's focus is not on theory. You can also choose any relevant ML application. It does not matter if it is deep learning, linear models, random forests, reinforcement learning or something else.

In other words, your report should look like **the "Experiments"-Section of an ICML, NIPS or ICLR machine learning paper**. You are not asked to invent new algorithms or convergence theory, but only to extend knowledge and insights about their experimental performance and behaviour, and possibly interpret this in the light of existing theory.

Grading. The project is done in groups of 3 students. It will count 30% to the final grade. Project reports (3 page PDF) and code are due June 13th.

Topic Examples

The list below is by no means exhaustive. You are encouraged to freely chose very different aspects and topics as long as they concern optimization and as long as you study them for any ML-relevant application.

- Local minima for deep learning: Can you find differences between the 'shape' of local minima that different optimizers finds, depending on different step-sizes or mini-batch size, vs e.g. Adam or SGD or full gradient descent?
- How does the order in which we see the training data influence the result?
- same above two questions for LLMs, GANs, or for matrix factorizations?
- Along a training trajectory for a deep net, does the behaviour of each step resemble the convex case, is it different early or late in training? Are the quantities used in our theory (such as noise bound, Lipschitzness etc) predictive or not of the actual trajectory?
- How do different optimization variants affect generalization (test error)?
- Second-order methods: Do (Quasi-)Newton methods go to differently shaped local minima in neural networks? Or: Is the secant method a viable alternative training method?
- Practical second-order ideas in deep learning: E.g. pre-conditioning as in shampoo, or AdaHessian.
- Quantized SGD: As quantization of gradients or iterates becomes coarser, what happens to the optimization algorithms, and to generalization? Is it different for DL as compared to linear ML models?
- Adam / signSGD / Lion / AdaHessian: Can you suggest/try different data-dependent coordinate-wise learning rate schemes and compare them?

- For training deep nets with very large SGD mini-batches: when does the scalability gain disappear? Is it influenced by other properties of the optimizer? For example, what is the effect of first slowly growing the learning rate and then later decreasing it?
- Meta-Learning: Can you learn the learning rate? The importance of each datapoint? The direction or curvature?
- How well do zero-order optimization methods do for ML applications, compared to standard first-order methods?
- Asynchronous SGD: How do different delays affect convergence? How does it interplay with momentum? Does it act as a regularizer, like drop-out?
- Some of the theoretical behaviors we analyzed in theory: Do they actually appear in practice or not, for convex models or deep nets?
- Frank-Wolfe or Coordinate-Descent for deep learning?
- Implicit regularization effect of SGD, as in flattening the landscape, or encouraging different trajectories or converging to solutions with different properties
- Federated learning or decentralized learning with different data distributions per participant (non-iid per participant).
- Sequential learning (life-long learning) when tasks change over time, on the level of data points or groups or clients/tasks participating
- Distributed or Decentralized SGD: How does the data heterogeneity between clients or the communication topology influence practical convergence? (experiments can be simulated)
- ...

Again, these areas are just mentioned for illustration, and you're very welcome to chose sth very different not featured on this list. The only aim is that it should be related to an optimization algorithm behavior on an application broadly related to ML.

Code pointers

- LLM training code example - github.com/epfml/llm-baselines
- How to compute Hessian-vector products? - iclr-blogposts.github.io/2024/blog/bench-hvp/

Deliverables

- **Written Report.** You will write a maximum 3 page PDF report on your findings, using LaTeX. You can use unlimited addition pages for references and for an appendix. The main paper should be self-contained!
- **Code.** In a language of your choice. Python with PyTorch is recommended for convenient access to gradients and optimizers. External libraries and existing implementations are allowed, if properly cited. You submit the complete executable and documented code, as a `github` repository link (make sure it's accessible). Rules for the code part:
  - *Reproducibility:* In your submission, provide a script like `run.py` or notebook that produces all results and plots you use in the paper.
  - *Documentation:* Your system must be clearly described in your PDF report and also well-documented in the code itself. A useful `ReadMe` file must be provided.

Submission URL: `https://mlcourse.epfl.ch/`

Submission deadline: June 12th, 2026 (16:00)

Grading Criteria

For those of you who have taken ML CS433, the grading criteria will be following the same principles.

We will grade you on the scientific contribution you made, that is on the insights gained compared to standard baseline methods. This is only possible based on a rigorous and fair empirical comparison. The criteria are

- **Solid comparison baselines supporting your claims**
  Quantify the benefits of your method by providing clear quality measurements of the most important aspects and additions you chose for your model. Start with a very basic baseline, and demonstrate what improvements your contributions yield.

- **Reproducibility**
  Your readers should be able to reproduce your results based on your report only. Describe what what hyper-parameters you selected and why, chose realistic and representative datasets, and clearly describe the overall pipeline you used.

- **Scientific novelty and creativity**
  You will likely be using more than the standard methods we saw in the course. Make sure that your report addresses the following points.

  - What is the *specific* aspect which you study, and why this is interesting and important.
  - Search for related work. Have similar experiments appeared in the literature? If yes, how are your experiments adding additional insight? Discuss the pros/cons of the existing studies compared to your approach.
  - How is the algorithm variant or the aspect of your choice helping for optimization speed, accuracy or generalization? For example, you should compare the optimization error with and without your object of study.

- **Writeup quality**
  Some advice when writing a scientific report:

  - Try to convey a clear story giving the most relevant aspects of your findings. Learning what has not worked can additionally help the reader (and help them better understand *why* you have made the many choices you did), but focus on what is most relevant and interesting.
  - Before the submission, have an external person proofread your report. Use a spell-checker.
  - Plots are an excellent way to share information that might be hard to convey by writing. Your plots should be understandable, have axis labels, appropriate axis ranges, and a self-contained caption.

As usual, we will automatically check your code and report for plagiarism.

Report Guidelines

Clearly describe your used methods, state your conclusions, and argue that the results you obtained make (or do not make) sense and the reasons behind it. Keep the report short and to the point, with a strict limit of 3 pages. References and an additional technical appendix are allowed. Those *do not* count towards this page limit, but do not expect the reviewer to read the appendix.

Use this LaTeXtemplate to get started with the report:

github.com/epfml/OptML_course/tree/master/labs/mini-project/latex-example-paper

The file also contains some more helpful information on how to write a scientific report or paper. We will also help you during the exercise session and office hours if you ask us.

For more guidelines on what makes a good report, see the grading criteria above. In particular, don't forget to take care about

- *Reproducibility:* Not only in the code, but also in the report, do include complete details about each algorithm you tried, e.g. what lambda values you used for ridge regression?

- *Baselines:* Give clear experimental evidence: It is crucial to report relative differences in the evaluation metrics, that is, with and without the element you study. Compare your approach to several properly implemented baseline algorithms.

Some additional resources on LaTeX:

- `https://github.com/VoLuong/Begin-Latex-in-minutes` - getting started with LaTeX
- `http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/` - tutorial on LaTeX
- `http://www.stdout.org/~winston/latex/latexsheet-a4.pdf` - cheat sheet collecting most of all useful commands in LaTeX
- `http://en.wikibooks.org/wiki/LaTeX` - detailed tutorial on LaTeX

Producing figures for LaTeX in Python

There are some good visualization tools in Python. “matplotlib” is probably the single most used Python package for 2D-graphics. The relevant tutorials are as follow:

- Matplotlib tutorial: `https://github.com/rougier/matplotlib-tutorial/`
- Matplotlib tutorial: `https://sites.google.com/site/scigraphs/tutorial`
- Matplotlib Tutorial: `http://jakevdp.github.io/mpl_tutorial/`

Regarding other useful Python data visualization libraries, please refer to this blog for more information.