Assignment 2 - Hacking

Due 19th December 2023

OVERVIEW AND GOALS

In this exercise, you will deep dive into your project, try to build a baseline model that operates on your data as one would expect (although not necessarily in an ideal way). Then try to optimize your neural network as much as you can (given your available time budget), i.e., perform hyperparameter optimization and/or gather more data.

N.B. If you picked the project type "Bring your own data," you still have to build and train a suitable pipeline, even if it is a very simple one. The results will then serve as a baseline for your newly created dataset.

RECOMMENDED APPROACH

1. Specify an error metric (how you want to evaluate your algorithm) and a reasonable target value for that error metric (how good it should be)
2. Establish a working end-to-end pipeline as a baseline for further experimentation
3. Instrument the system to detect defects or bottlenecks that cause underfitting/overfitting
4. Repeatedly make incremental changes, i.e., gather new data, adjust hyperparameters, change algorithm depending on new insights

Be aware that hyperparameter optimization can be costly and time-consuming, so start early and stop if you've reached your own target. Gathering new data can be more useful than you think and there are many ways to artificially expand your dataset at low costs (= data augmentation).

Whenever you work on your project, please keep track of the time to help you better understand which parts of the project potentially require more time than you initially estimated.

DELIVERABLES

For this exercise, you are expected to implement the project you specified in exercise 1. Changes should be documented in your Git repository. Please also provide a brief summary (e.g., in your README) of

- the error metric you specified
- the target of that error metric that you want to achieve
- the actually achieved value of that metric
- the amount of time you spent on each task, according to your own work breakdown structure.

Keep in mind that the actual amount of time you spend on your exercise does not affect grading!

As this is a software project, students are expected to adhere to the principles of good software design, which include:

- Use of good names for your variables, methods, functions, and modules. These names should be self-explanatory, easy to understand, and intention-revealing. They may be short if they are confined in a small scope (e.g., 'i' for a counter in a loop) but should be long enough to grasp their intended use (e.g., 'input_image', 'number_of_epochs', 'training_batch_size'). Avoid abbreviations because they are hard to understand. The only exception to this rule is if the abbreviation is better known than it's spelled out counterpart (e.g., 'HTML', 'XML', 'HTTP').
- Providing as much documentation as necessary (useful documentation includes architectural diagrams or comments in the code that explain *why* a certain operation was implemented in a specific way). Avoid useless comments that simply repeat what is already in the code (e.g., 'Computes the scaling factor' for a method called 'compute_scaling_factor'). A great way to create documentation directly from the source code is by using Read the docs or similar tools.
- Test your code. Create at least a few tests that make sure your pre-processing and post-processing work correctly. Testing the training procedure itself is difficult and therefore not mandatory (but you will receive bonus points if they are done, e.g., train on the CPU of the CI with a tiny fraction for a single epoch). Ideally, you write the tests before the actual implementation (Test-Driven-Development).
- Make sure your application can be run easily, document how to do so, and ensure that the tests are green. A great way to do this is by using a Continuous Integration server that automatically builds your application and runs the tests on each commit. There are plenty of free services available, including Travis, Gitlab, AppVeyor, CircleCI, Bitbucket Pipelines, Azure DevOps, or Bitrise (depending on your platform and programming language). Setting up a CI in the beginning can seem like unnecessary overhead, but it will help you significantly in the long run.
- If you use Python, please create a requirements.txt file that specifies the used libraries, such that they can be installed easily in a fresh virtual environment.
- Please auto-format your code before you submit it. All major IDEs offer this functionality.

There are several platforms that can help you execute and log your experiments, including Google Colab, Weights and Biases or Gradient.

Once you're done with your work, notify me by writing an e-mail to alexander.pacha@tuwien.ac.at with the subject **[Applied Deep Learning] Exercise 2 - Your Matr.Number**. Since you've already given me access to your repository, no more action is needed.

CRITERIA FOR GRADING

As discussed in the preliminary lecture, every assignment will be graded according to these five criteria (as applicable) with a maximum score of 10 points:

- Results
- Creativity
- Complexity
- Code Quality
- Presentation

Late Policy

Submissions will still be accepted after the deadline, but keep in mind that there is a penalty of one point per day. So if you submit your assignment two days late, you will only be able to achieve a maximum of 8 points.

OTHER QUESTIONS

In case you have other questions regarding the assignment, please send an e-mail to alexander.pacha@tuwien.ac.at with the subject line starting with **[Applied Deep Learning]** or post your question in the discussion forum in RADIX/TUWEL (preferred).