To get access to this week's code use the following link: https://classroom.github.com/a/YHwgOJ7k

---

General constraints for submissions: Please adhere to these rules to make our and your life easier! We will deduct points if you fail to do so.

- Your code should work with *Python 3.8*.
- You should only fill out the *TODO-gaps* and not change anything else in the code.
- Add comments to your code, to help us understand your solution.
- Your code should adhere to the PEP8 style guide. We allow line lengths of up to 120.
- While working on the exercise, push all commits to the `dev` branch (details in assignment 1). Only push your final results to the `main` branch, where they will be automatically tested in the cloud. If you push to `main` more than 3 times per exercise, we will deduct points.
- All provided unit tests have to pass: In your *GitHub* repository navigate to *Actions* → your last commit → Autograding → education/autograding to see which tests have passed. The points in autograding only show the number of tests passed and have nothing to do with the points you get for the exercise.
- `for` loops can be slow in Python, use vectorized `numpy` operations wherever possible (see assignment 1 for an example).
- Submit *a single PDF* named `submission.pdf`. Include your matriculation numbers on the top of the sheet. Add the answers and solution paths to all non-coding questions in the exercise. Do not leave answers to any questions as comments in the code. You can use Latex with the student template (provided in exercise 1 / ILIAS) or do it by hand.
- Please help us to improve the exercises by filling out and submitting the `feedback.md` file.
- We do not tolerate plagiarism. If you copy from other teams or the internet, you will get 0 points. Further action will be taken against repeat offenders!
- Passing the exercises ($\geq 50\%$ in total) is a requirement for passing the course.

How to run the exercise and tests

- See the `setup.pdf` in exercise 1 / ILIAS for installation details.
- We always assume you run commands in the *root folder* of the exercise repository.
- If you use miniconda, do not forget to activate your environment with `conda activate mydlenv`
- Install the required packages with `pip install -r requirements.txt`
- Python files in the *root folder* of the repository contain the scripts to run the code.
- Python files in the `tests/` folder of the repository contain the tests that will be used to check your solution.
- Test everything at once with `python -m pytest`
- Run a single test with `python -m tests.test_something` (replace `something` with the test's name).
- To check your solution for the correct code style, run `pycodestyle --max-line-length 120 .`
- The scripts `runtests.sh` (Linux/Mac) or `runtests.bat` (Windows) can be used to run all the tests described above. If you are on Linux, you need execution rights to run `runtests.sh`.

This exercise focuses on regularization in optimization. We will:

- Implement Dropout.
- Implement L1/L2 loss.
- Analyze the effect of the different regularization methods on the parameter distribution.
- Think about early stopping and data augmentation.

1. Coding Tasks

In this part of the exercise we will take a look at multiple regularizers.

1) [3 points] **Todo:** Implement the Dropout module (class `Dropout` in file `lib/regularizers.py`).

During training, the dropout layer randomly sets elements of the input tensor to 0 with probability p_{delete} and scales the remaining values by $\frac{1}{1-p_{delete}}$. During evaluation, the dropout layer returns the identity.

Note: In the slides and the deep learning book, p describes the probability of an input being *retained*, while in this exercise (and PyTorch), $p_{delete} = 1 - p$ describes the probability of an input being *zeroed*.

Note: In the slides and the deep learning book, the network weights are *downscaled* by p during *inference*, while in this exercise (and PyTorch) the weights are *upscaled* by $\frac{1}{1 - p_{delete}}$ during *training*.

Run `python -m tests.test_dropout` to test your implementation.

2) [4 points] **Todo:** Implement L1/L2 Regularization (class `L1Regularization` and `L2Regularization` in `lib/regularizers.py`)

   This is one of the rare cases where we deviate from the pytorch API. The reason is that pytorch implements L_2 regularization in the optimizer but calls it 'weight_decay' (which is actually a different operation if you're not using SGD). You can read more on this here.

   Run `python -m tests.test_L1_regularizer` and `python -m tests.test_L2_regularizer` to test your implementation.

3) [3 points] **Todo:** Train the four models on MNIST. Complete the model building functions `build_model` and `build_model_dropout` as well as function `train_models` in file `lib/experiments.py`. Please note that we implemented `RegularizedCrossEntropy` in `lib/regularizers.py` for you, which will help you in parts of the exercise.

   To visualize the effect of regularization on the model parameters, you'll train the following models:

   - Without regularization. It's always a good idea to train your model without any regularization first. Not only to have a baseline but also to check if your model is able to overfit the training data. If it can't overfit, it's likely not powerful enough or there's an error in the implementation. We already include the training of this model as an example.
   - With $L2$ regularization, $\lambda = 0.0001$, on the weight parameters (not on the bias).
   - With $L1$ regularization, $\lambda = 0.0001$, on the weight parameters (not on the bias).
   - With Dropout, $p_{delete} = 0.1$ for the input, $p_{delete} = 0.2$ for the hidden layers.

   Run `train_models.py` to run your code.

4) [2 points] **Todo:** Plot the parameter distribution (function `plot_distribution` in `lib/plot.py`).

   Compare the parameter distribution of the five models by plotting the histogram of their weight values from -1 to 1 with 100 bins. Use matplotlib.pyplot.hist for plotting the histogram.

   Run `plot_distribution.py` to see the plot. Include the resulting figure in `submission.pdf` and briefly discuss your observations.

2. Pen & Paper Questions

1) [2 points] **Data Augmentation**
   Data augmentation increases model generalization by augmenting the training set with fake data.

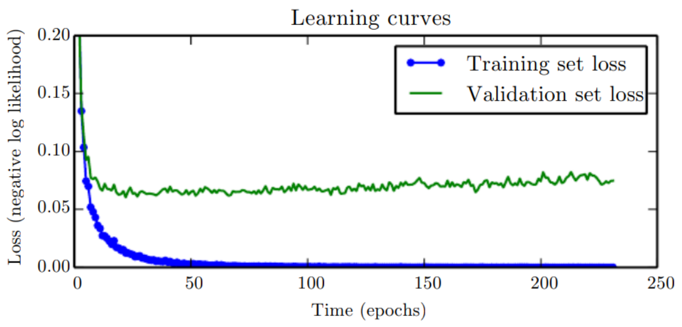
   1. State five effective operations to generate fake data on the MNIST dataset.

2. State one operation which doesn't make sense and why.

2) [2 points] **Early Stopping**
Another very popular technique in deep learning is early stopping (Deep Learning Book, section 7.8).

1. How do the given loss curves (Deep Learning Book, figure 7.3) relate to early stopping?

2. Why is it a regularization technique?



3. [1 bonus point] **Code Style**

On each exercise sheet, we will also be using `pycodestyle` to adhere to a common Python standard. `pycodestyle` checks your Python code against some common style conventions in PEP 8 and reports any deviations with specific error codes.

Your code will be automatically evaluated on submission (on push to `main`). You can run `pycodestyle --max-line-length=120 .` to manually evaluate your code before submission.

One bonus point will be awarded if there are no code style errors detected in your code by the `pycodestyle --max-line-length=120 .` command.

4. [1 bonus point] **Feedback**

Todo: Please give us feedback by filling out the `feedback.md` file.

- Major Problems?
- Helpful?
- Duration (hours)? For this, please follow the instructions in the `feedback.md` file.
- Other feedback?

This assignment is due on 27.11.2024 23:59 CET. Submit your solution for the tasks by uploading (`git push`) the PDF, txt file(s) and your code to your group's repository. The PDF has to include the name of the submitter(s). Teams of at most 3 students are allowed.