To get access to this week's code use the following link: https://classroom.github.com/a/REwuLU3q

---

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 convolutional neural networks. We will:

- Implement a convolution layer
- Extend the convolution with various parameters
- Apply a convolutional network

1. Pen and Paper Tasks

To get an intuition of how convolution works, you are going to solve the forward as well as the backward pass of a 1D cross-correlation of a given input. Remember that convolution is just like cross-correlation, except that we flip over the filter before correlating.

You are given the following:

An input, $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

A kernel, $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$

A scalar bias, b.

When this kernel is applied by sliding over the given input, with a stride of 1 and no padding, we expect an output
$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$

Consider the loss function L to be given by:

$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$

Remember, do not flip the kernel, since we compute cross-correlation.

Your tasks are:

1) [1 point] (core) Perform a single forward pass of this cross-correlation with stride 1 and no padding. Then, plug in the following values for $\mathbf{x}, \mathbf{w}$ and b to get $\hat{\mathbf{y}}$:

   $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$

   $\mathbf{w} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$

   $b = 1$

   Make sure you show the formulas, not just the end results. Also, compute the loss L. Consider $\mathbf{y} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$ to be the ground truth.

2) [2 points] Perform a single backward pass of this convolution, i.e., find $\frac{\partial L}{\partial \mathbf{w}}$ and $\frac{\partial L}{\partial b}$.

3) [1 point] Now, plug in the values from (1) above to get the updated values of $\mathbf{w}$ and b at the end of the backward pass. Assume a learning rate of 0.01. Perform one more forward pass with the updated weights and bias. What is the loss now?

2. Image Dataset

With convolutions, we finally can treat the MNIST data samples as what they are: grayscale images. Instead of flattened features, we load them in 3D shape (number of channels, height, width). The number of channels in this case is 1. When we consider the different samples too, the shape of `x_train` now becomes:

`x_train.shape == (n_samples, channels, width, height) == (50000, 1, 28, 28)`.

Run file `plot_mnist.py` to see how the samples look.

3. [1 point] Convolution Output Size (core)

The output of the convolution operation depends on the input size (I), filter size (W), stride (S), and padding (P).

Todo: Implement the formula for computing the output dimensions of a convolution operation. Complete the function `calculate_conv_output` in file `lib/convolutions.py` and run file `show_conv_output_dims.py`

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

4. ConvLayer Implementation

1) [2 points] (core)

   In this exercise, you have to implement the forward pass of your own convolutional layer, which actually computes *cross-correlation* (this is to ensure compatibility with PyTorch). **This means we do not flip the kernel.**

   But before you dive in, finish the TODOs in `lib/convolution_warmup.py`. This exercise will familiarize you with the building blocks of the forward pass of a 2D convolution. You can then apply what you learned in the next part of this exercise.

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

2) [2 points] (core) Having finished the warmup, you're now ready to implement a forward pass of the 2D convolution. The backward pass has already been implemented for you.

   Input dimensions for the 2d conv layer are `(batch, channel, width, height)`.

   Todo: Complete class `Conv2D` in file `lib/convolutions.py` and run file `run_simple_conv.py`

   Note: Work iteratively. E.g. first implement simple convolution, add stride and padding afterwards. Additionally, note that the tests in `tests/test_conv_layer.py` are also iterative.

   Note: You may iterate over that last two axes of the activation map using nested for-loops. However, to keep the implementation fast, **do not use more than two nested for-loops in the convolution**. Broadcasting over channels can be helpful here.

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

5. [2 points] Experiments (core)

Now let us finally train our model. Make sure to set the correct input dimensions. For faster training, we have limited the number of samples to `10000`. You can reduce the number of samples to train/validate on if training takes too long on your device, or increase it to see if results improve. To do so, use the argument `--max_datapoints NUM_DATAPOINTS` when running either of the two experiment scripts.

Todo: Complete function `create_conv_model` in file `lib/experiments.py` and run file `run_conv_experiment.py`.

You should get roughly 96% accuracy at the end of the tenth epoch.

6. Equivariance

1) [2 points] Implementation

Let us now explore an important property of convolutions: equivariance. For this, we will shift the validation data by two pixels to the right and bottom. Then, we will compare the accuracies of a standard MLP and the convolution model we trained above on the shifted data.

Todo: Complete functions `run_shifted_conv_experiment` and `shift_data` in file `lib/experiments.py` and run the file `run_shifted_conv_experiment.py`

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

2) [3 points] Answer the following questions (include your answers in **submission.pdf**)

Todo: What do you observe regarding the accuracies? How can the results be explained?

Todo: What could we do to improve the accuracy of the MLP model on the shifted validation set?

Todo: Why is the performance of the CNN model not equal for unshifted and shifted data, if convolutions are equivariant to translation?

7. [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.

8. [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 04.12.2024 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.