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

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 `master` branch, where they will be automatically tested in the cloud. If you push to `master` 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` with the answers and solution paths to all pen and paper questions in the exercise. 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 ($\geq 50\%$) 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.

This exercise focuses on probabilities, autoencoders and GANs. In the first part we will recap basics of probability theory. In the second part, we're going to implement an autoencoder and explore the learned latent space.

1. [2 points] **Probability Theory: Bertrand's Box Paradox**

Solve mathematically. Please provide intermediate steps and explanations.

Consider a box containing the following 3 cards:

- one with two black sides
- one with two white sides
- one with a black and a white side

From the three cards, one is drawn at random and put on the table. You can only see the side facing up.

Todo: Answer the following questions in your `submission.pdf`:

1. What are the probabilities that the card on the table shows a black side? What are the probabilities it shows a white side?

2. If we draw a card and it shows black, compute the probability that the other side of the card is also black.

3. Find the the probability that the other side of the card is black if the card shows a white side.

2. **Distributions and the Central Limit Theorem**

1) [1 point] The central limit theorem states that for i.i.d. random samples $\{X_i\}$ from an (almost) arbitrary distribution with given mean μ and variance σ^2, the mean $\frac{1}{n}\sum_{i=1}^{n} X_i$ follows approximately a normal distribution. More precisely, it reads

$$\frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{n\to\infty} \mathcal{N}(\mu, \frac{\sigma^2}{n}) \,.$$

Todo: Complete function `plot_clt` in file `lib/distributions.py` and run file `plot_clt.py`.

Draw $n = 1, 16, 64, 1024$ samples from the distributions below for 1024 times (each). Draw for each (n, distribution) pair a histogram over the sample mean. Include the corresponding normal distribution (pdf) in the plot.

- the exponential distribution $p(X) = \lambda e^{-\lambda X}$ with $\lambda = 1$
- the Gaussian/normal distribution with $\mu = 1, \sigma = 1$

2) [2 points] Now assume that you can only sample from uniform distributions. Implement functions to sample from an approximated standard normal distribution and an approximated normal distribution. Plot the distributions in comparison to the numpy implementation.

Todo: Complete functions `std_normal` and `normal` in file `lib/distributions.py` and run file `plot_normal.py`

Hint: Use the Central Limit Theorem: Draw $N * M$ samples from a uniform distribution and calculate the N means of those samples. Those means then approximate a norm distribution.

The variance of a uniform distribution $U(a, b)$ is $\sigma_u^2 = \frac{1}{12}(b-a)^2$

So for a uniform distribution $U(-b, b)$ the variance is $\sigma_u^2 = \frac{1}{12}(2b)^2 = \frac{1}{3}b^2$

The Central Limit Theorem states that the means of a number of samples from this uniform distribution will follow a normal distribution with the same mean as the uniform distribution and variance $\sigma_n^2 = \frac{\sigma_u^2}{n_u}$ where n_u is the number of samples.

Given that we want to sample from a standard normal distribution we know that $\sigma_n = 1$.

Solve for b to know from which uniform distribution you need to sample in the task above.

3. **Variational Autoencoder**

Let's now implement our own variational auto encoder (VAE). The VAE is described in chapter 20.10.3 of the Deep Learning Book and you can find the original paper here.

Compared to a "standard" auto encoder, a VAE learns to approximate the posterior.

Check the file `vaeloss_derivation.pdf` for the derivation of the loss formula. You will need the result of this derivation to implement the loss.

1) [4 points] Implement the VAE.

Todo: Fill the TODO-gaps in file `lib/model_vae.py` class `VAE` and in file `lib/loss_vae.py` class `VAELoss`. Run file `run_vae_training.py` and check the output images in folder `results_kl1.0/`

Compare them to our provided VAE output file `vae_output_15_epochs.png`.

2) [2 points] Exploring the latent space.

One of the easiest ways to visualize the latent space is to limit the size to two dimensions (which of course might not always capture the data well) and to sample from the model over a 2D grid. This is what we're going to do now.

Also, we can nicely visualize the concept of a VAE. We plot the mean over the estimated means for each class (the numbers 0 to 9) and the mean estimated standard deviation for each class (the blue crosses).

Todo: Fill the TODO-gaps in file `lib/explore_latent_space.py` functions `get_mu_logvar` and `sample_on_grid`. Run file `plot_latent_space.py`.

3) [2 points] Exploring the influence of the KL-divergence on the loss.

Let's now investigate the influence of the KL-divergence by training:

- A model where we weight the KL-divergence part of the loss by a factor of 30.
- A model where we remove it (weight 0).

Then we visualize the latent space as we did above.

Todo: Run the training and plotting script and set the `--kl_loss_weight` argument. Note that the output folder will be `results_kl#/` where # is the KL divergence loss weight.

- `python run_vae_training.py --kl_loss_weight 30`
- `python plot_latent_space.py --kl_loss_weight 30`
- `python run_vae_training.py --kl_loss_weight 0`
- `python plot_latent_space.py --kl_loss_weight 0`

Todo: Answer the following questions in your `submission.pdf`:

- What do you observe?
- How can these results be explained?
- What is the role of the KL divergence term?

4. [3 points] **Generative Adversarial Networks**

GANs can create sharper images than VAEs, so for purely generative tasks they are usually preferred over VAEs. For more information, see the file `assignment_gan.pdf`.

This exercise is an adaptation of the PyTorch tutorial by Nathan Inkawhich for training a GAN on the Celeb-A Faces Dataset. In this experiment, we will be using the flowers dataset from the competition instead. A downscaled version of the images are provided in folder `dataset_flowers_64px`.

Note: Training on the flowers dataset for the default 500 epochs will take about 20–30 minutes on a GPU and about 1–2 hours on a CPU. You can also run the GAN training on the GPU of one of the TF pool computers as in the competition.

Todo: Your goal is to fill the TODO gaps in `lib/model_gan.py`. The docstrings of the Generator and Discriminator explain how the final model should look like.

Todo: Run `run_gan_training.py` and look into the folder `results_gan` to see your experiment results.

We provide model output after 500 and 2500 epochs of training for you to compare to your own results.

Optional: If you are not happy with the results, you could change the hyperparameters. Some suggestions:

- Train for more epochs.
- Increase model capacity (feature map sizes ngf, ndf and size of latent vector nz).
- Change the batch size.

We also provide the generator model checkpoint after 2500 epochs. You could *optionally* write your own code to load the checkpoint and generate some more results. To do that, sample from a normal distribution with mean 0, variance T where the default temperature $T = 1$, reshape to (batch_size, latent_size = 64, 1, 1), feed the sample into the generator, run the output through the `batch_images_to_numpy` function and use `plt.imsave` function to save the results to a file.

5. [1 bonus point] **Code Style**

On every exercise sheet, we will also make use of `pycodestyle` to adhere to a common python standard. Your code will be automatically evaluated on submission (on push to master). Run `pycodestyle --max-line-length=120 .` to check your code locally.

6. [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 27.01.2022 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.