To get access to this week's code use the following link: https://classroom.github.com/a/1PQVd1co

---

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`.

Important: For this assignment you should either go back to using the environment from assignment 09 "Practical" or create a new environment, and install the requirements.txt and the ffmpeg package:

```
>>> conda create --name new_env python=3.8
>>> conda activate new_env
>>> pip install -r requirements.txt >>> conda install -c conda-forge ffmpeg
```

In this assignment we will learn how Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) differ in:

- the way we can train them to generate new data.
- the way we can then generate new data from them.
- the appearance of the generated data.

We will see that results from GANs look more realistic and sharper, whereas for VAEs we can actively influence the appearance of the generated data by sampling from a certain region of the latent space.

1. Variational Autoencoder

First, let's implement our own variational autoencoder (VAE). The VAE is described in chapter 20.10.3 of the Deep Learning Book and you can find the original paper here.

Whereas a standard auto encoder learns a (deterministic) mapping from the input to a representation in the latent space, a VAE learns a posterior distribution over the latent space given the input.

1) [4 points] (core) Implement the VAE.

   In this task we will learn how to implement the VAE architecture with an encoder and decoder network. We will also see how to implement the reparameterization trick in order to be able to backpropagate through the step of sampling from the latent space.

   Todo: Fill the TODO-gaps in file `lib/model_vae.py` class `VAE`.

2) [2 points] Implement the VAELoss

   As part of the VAELoss we need to compute the KL-divergence between two normal distributions. In `vaeloss_derivation.pdf` we provide the derivation for it.

   Fill the TODO-gaps in file `lib/loss_vae.py` class `VAELoss`.

   Having implemented the VAE and the VAELoss we can now train the VAE. Run file `run_vae_training.py` and check the output images in folder `results_kl1.0/`

   Your results should look similar to the provided VAE output file (`vae_output_15_epochs.png`).

3) [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`.

4) [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?

2. Diffusion Models

Diffusion Models define a Markov chain of diffusion steps to gradually introduce random noise to data. They then train a neural network to reverse the diffusion process and reconstruct desired data samples from the noise. Diffusion models overcome several disadvantages of GANs, such as model collapse, vanishing gradients, and failure to converge, while generating images that are qualitatively and visually appealing.

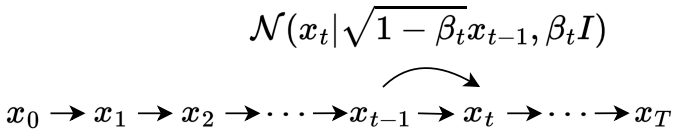

Figure 1: Forward Diffusion Process

1) [3 points] (pen and paper) In the lecture we have seen that a forward diffusion process as described in figure 1 is defined by the following transition probabilities. Let the schedule be $\{\beta_t \in (0,1)\}_{t=1}^{T}$. Let $\alpha_t = 1 - \beta_t$, then

$$\begin{aligned} q(x_t|x_{t-1}) &= \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1}, \beta_t I) \\ &= \mathcal{N}(x_t|\sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I) \end{aligned}$$

Hence given x_{t-1} we can represent x_t, where $\epsilon_{t-1} \sim \mathcal{N}(\epsilon|0, I)$, as:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

- Consider a scenario where the length of the diffusion process is three. Derive a closed form solution to $q(x_3|x_0)$ using the assumptions above. Precisely derive the mean μ and variance $\mathbf{\Sigma}$ of this Gaussian distribution. Please include all the necessary steps and justifications for every step. *Hint: Use property of sum of Gaussians and use substitution*

$$q(x_3|x_0) = \mathcal{N}(x_3|\mu x_0, \mathbf{\Sigma} I)$$

- How does this derivation generalize to a closed form solution to $q(x_t|x_0)$? Please include all the necessary steps and justifications for every step.

2) [3 points] (code) In this exercise you will write a small diffusion model to estimate different 2-d densities.

Todo: Your task is to complete the TODO blocks in `lib/utils.py` and `lib/train_diffusion.py` according to the instructions in the code. Refer to `lib/model_diffusion.py` to get a look into the diffusion model instantiated.

Todo: Run `run_diffusion_training.py` to train the diffusion model to estimate the underlying density. A video of the forward diffusion will be saved in your working directly.

Optional: Try the following :

- Increase number of diffusion steps
- Increase the number of epochs
- Switch to different possible densities/datasets

3. Generative Adversarial Networks *(Optional)*

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`. *Note that this exercise is optional and only for learning purposes* 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`.

1) [0 points] (core) Implement a GAN

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.

2) [0 points] (core) Implement the alternating training scheme of Discriminator and Generator in a GAN

Todo: Your goal is to fill the TODO gaps in `lib/train_gan.py`. The docstrings of the functions update_generator and update_discriminator explain how to implement the alternating training scheme in a GAN.

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

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 (for example approximately 1 hour on an Intel i5 processor and 8GB RAM).

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

Optional: If you are not happy with the results (and have enough compute available) 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.

4. [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.

5. [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 16.01.2024 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.