Due: 29.01.25 (23:59 CET) | **Deep Learning** | S. Adriaensen & A. Valada
Points: 16 + 5 bonus | 11th Assignment | WS 2024/2025

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

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

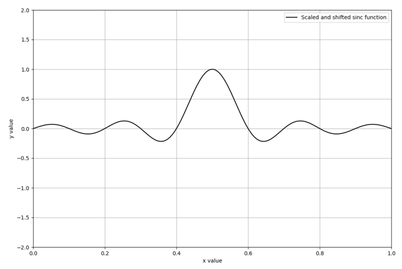
- See the `setup.pdf` in exercise 1 / ILIAS for installation details.
- We always assume you run commands in the *root folder* of the exercise repository.
- If you use miniconda, do not forget to activate your environment with `conda activate mydlenv`
- Install the required packages with `pip install -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`.

In this exercise we will investigate different approaches to express uncertainty over the predictions of our Neural Networks: DNGOs, which use Bayesian Linear Regression in their last layer, Ensembles, and as a bonus task Prior Fitted Networks. We will also get some hands-on experience with an important basic concept, that enables us to sample from distributions that we can't compute in closed form: Markov Chain Monte Carlo.

1. Uncertainties using DNGO

1) [4 points] DNGO Implementation

In this first part of the exercise we will be using NN models to fit the rescaled and shifted version of the sinc function (shown below).



As stated in the lecture, DNGO first trains a NN to completion and then takes the learned features from the last hidden layer and performs Bayesian Linear Regression (BLR) on it.

An implementation of BLR is already provided for you in `lib/blr.py`. The implementation contains a method to calculate the posterior distribution of the weights given the data and a method to calculate the posterior predictive distribution. You can use this already implemented class to perform the BLR part of DNGO.

Todo: Complete the `fit_blr_model` and `predict_mean_and_std` methods of the DNGO class in `lib/model.py`

Note: We also want to learn the bias. We achieve this in our BLR code by treating the bias as a weight and padding each input (here, the learned features) with 1 since $\boldsymbol{w} \cdot \boldsymbol{x} + b = [\boldsymbol{w}, b] \cdot [\boldsymbol{x}, 1]$.

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

Now we can train the base NN for DNGO. The training loop is already implemented for you.

Todo: Run the `run_dngo.py` file.

2) [3 points] **Interpreting Uncertainty Predictions**

Now we will plot the uncertainty of the NN we have just trained.

Todo: Run `eval_dngo.py` in order to see the plots.

The first plot will show the true function and predictive functions of a vanilla neural network, the mean of DNGO as well as sampled networks from DNGO. The second plot shows the true function and the mean prediction of DNGO with the uncertainties at each point for different uncertainty bands.

Answer the following questions (include your answers in **submission.pdf**):

1. Are the predicted uncertainties reasonable? Mention (at least) one positive and one negative aspect. In your answer, clearly distinguish between the quality of uncertainty predictions, and the quality of the predicted value/mean.

2. What happens to the predicted uncertainty as we make predictions for points outside of the training data range (i.e. extrapolation)?

3. Assume our model would fit the training data perfectly.
   Would the predicted uncertainty for the training data be 0? Why (not)?

2. Uncertainties using Ensembles

1) [3 points] **Ensemble Implementation**

We will now train an ensemble of NNs and get the mean prediction as the mean over the predictions of the individual NNs and the std. deviation as the std. deviation over the predictions.

Todo: Complete the code for the EnsembleFeedForward class in `lib/model.py`

Note: This class holds a list of base NNs and has a method `predict_mean_and_std` to predict the ensemble mean and std. deviation for them. Additionally, it has a method to return the prediction of each individual base NN which we will use in next exercise for plotting.

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

Todo: Run `run_ensemble.py` to train an ensemble of 5 models and save the results.

Note: Each model in the ensemble has the same architecture, but is initialized differently by drawing weights from a normal distribution.

2) [3 points] **Plot Multiple Predictions**

Todo: Complete the plotting function `plot_multiple_predictions` in `lib/plot.py` to plot individual predictions of the members of the ensemble.

Todo: Run `eval_ensemble.py` to evaluate the models by plotting their joint uncertainty in addition with visualization of multiple individual predictions.

Answer the following question (include your answer in **submission.pdf**):

1. How do these results compare to those of DNGO? Is this comparison fair?

3. [3 points] MCMC sampling

Here we implement the metropolis hastings algorithm and use it to sample from 3 different probability distributions by only using their un-normalized probability density functions. The first and second distributions are 1d gaussian mixture models with 2, 3 components respectively. The last one is a 2d gaussian mixture model with 2 components.

Todo: Complete the `metropolis_hastings` method in `lib/mcmc.py`

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

Todo: Run `plot_mcmc.py` to visualize the sampled points from above distributions. You can play around with the `burn_in` parameter in the main function to visualize its impact on the generated samples.

Answer the following question (include your answer in **submission.pdf**):

1. How does the burn_in time affect the sampling process in general, and also specifically in the case of above 3 distributions?

Hint: Keep all the other parameters fixed while changing the `burn_in` value in orders of 10. You can also decrease `num_samples` for a more apparent effect of `burn_in` value on generated samples.

4. [3 bonus points] **Uncertainties using PFNs**

Prior Fitted Neural Networks (PFNs) were recently proposed as an alternative way of doing Bayesian inference. Here, instead of using an analytical approach such as Gaussian Processes (GPs), we learn Bayesian inference from data, training a transformer architecture on datasets sampled from the prior to directly predict the posterior predictive distribution.

Todo: Use this demo to experiment with PFNs and answer the following questions (include your answers in **submission.pdf**):

1. Compare PFNs to GPs on the following dataset:

x	**y**
0.2	0
0.4	-1
0.6	1
0.8	0

   What do you observe?

2. How is the dataset you just entered used during PFN training / inference?

3. What benefit do PFNs have over GPs (in general)?

4. Compare PFNs to GPs again, this time on the following dataset:

x	**y**
0.2	0
0.49	-1
0.51	1
0.8	0

   Explain your observations.

5. [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.

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 29.01.25 (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.